AFRL-ML-WP-TR-2001-4077

PARTS OBSOLESCENCE MANAGEMENT TOOLS (POMT)

Tom Regan

TRW
1900 Founders Drive
Kettering, OH 45420

MARCH 2001

FINAL REPORT FOR 15 APRIL 1998 – 04 FEBRUARY 2001

Approved for public release; distribution unlimited

MATERIALS AND MANUFACTURING DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7750

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASC/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

THEODORE FINNESSY
Project Engineer
Information Integration Team
Adv. Manufacturing Enterprise Branch

KENNETH A. FEESER
Leader
Information Integration Team
Adv. Manufacturing Enterprise Branch

BRENCH BODEN
Chief
Adv. Manufacturing Enterprise Branch
Manufacturing Technology Division

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2001	Final Report, 04/15/1998 – 02/04/2001

4. TITLE AND SUBTITLE
Parts Obsolescence Management Tools (POMT)

5. FUNDING NUMBERS
C: F33615-98-C-5129
PE: 78011F
PR: 2865
TA: 07
WU: 23

6. AUTHOR(S)
Tom Regan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TRW
1900 Founders Drive
Kettering, OH 45420

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
MATERIALS AND MANUFACTURING DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7750
POC: Theodore J. Finnessy, AFRL/MLLS, (937) 904-4344

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-ML-WP-TR-2001-4077

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The main focus of the Parts Obsolescence Management Tools (POMT) program was the definition of a reengineering methodology that supports both the initial design and the reengineering of modern products. This methodology begins with the product specification using the concept of the simulatable specification as defined by the Continuous Electronic Enhancements using Simulatable Specifications (CEENSS) program and continues with the reengineering process whereby an implementation is developed and culminates with the interface to the manufacturing process. This effort also focused on the development of two key computer-aided design (CAD) tools needed to support the application of this methodology in real-world product challenges. These tools include a behavioral product reengineering (BRP) tool and design verification test generator (DVTG) tool.

14. SUBJECT TERMS
reengineering

15. NUMBER OF PAGES
110

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Table of Contents

Table of Contents (cont.)

SECTION PAGE

Table of Contents (cont.)

List of Figures

List of Tables

1. INTRODUCTION

The main focus of the Parts Obsolescence Management Tools (POMT) program was the definition of a reengineering methodology that supports both the initial design and the reengineering of modern electronics products. This methodology begins with the product specification using the concept of the simulatable specification as defined by the Continuous Electronic ENhancements using Simulatable Specifications (CEENSS) program and continues with the reengineering process whereby an implementation is developed and culminates with the interface to the manufacturing process. This effort also focused on the development of two key computer-aided design (CAD) tools needed to support the application of this methodology in real-world product challenges. These tools include the Behavioral Product Reengineering (BPR) tool and a Design Verification Test Generator (DVTG) tool.

1.1 POMT Program

Parts obsolescence, also referred to as out-of production parts (OPP), is a major issue which must be faced both in the design of modern weapon systems as well as in the support of older legacy systems. This has become evident in TRW's design and planning for manufacturing of the communications, navigation and identification (CNI) systems for F-22 and Comanche helicopter. From the second quarter of 1995 through the second quarter of 1996, TRW experienced 62 OPP events across the total set of 693 active component line items. The 16 most serious of these events forced redesign. The expected continuation of this problem has driven TRW to adopt a periodic preplanned product improvement (P^4I) strategy that integrates elements of design, manufacturing and support. The TRW company's strategy preempts obsolete parts problems through preplanned periodic redesigns of the electronics, the recurring costs of which are recovered through savings in production and support. In addition, the periodic incorporation of new technology with higher processing densities allows for performance enhancements and savings in weight and cost. Using this strategy since 1995, TRW has improved the F-22 CNI functionality and durability while reducing the total number of modules from 16 to 6, thereby reducing weight and costs. Contributing significantly to the success of this strategy is the aggressive adoption of commercial business practices and the move towards contractor logistics support (CLS).

The TRW company has adopted the P^4I strategy for its digital and receiver product road maps, through which F-22 designs are being upgraded and migrated to the Comanche helicopter and Joint Strike Fighter (JSF). Key technology components of these road maps include the movement toward the use of field programmable gate arrays (FPGAs) in lieu of application specific integrated circuits (ASICs), VHSIC hardware description language (VHDL) to capture both design requirements as well as behavior, and the use of commercial radio frequency (RF) microwave technology.

The TRW company's move to this P^4I strategy has been challenged by the lack of key methodologies and CAD tools. The prototype CAD tool set developed on the CEENSS program provided a good starting point, but needed to be expanded in the areas of behavioral product design synthesis and automation in the development of design verification tests. This program proposed to fill this shortfall through development of new tools that enhance the prototype CEENSS tool set by completing the commercialization of the formal requirements modeling environment and by adding two key additional tools. These tools include a BPR capability and DVTG. The BPR is built on top of the Synopsys Design Environment (SDE) tool and incorporates other key product design tools from Synopsys such as their Behavioral Compiler tool. The DVTG tool is based on a VHDL specification language (VSPEC), a formal requirements specification language developed by the University of Cincinnati. This language has evolved into the system level description language (SLDL). Formal requirements support tools were a key component of simulatable specifications capability demonstrated on the CEENSS program. These tools not only have relevance to modern systems, but to older legacy electronics systems as well, using the captured legacy design as a baseline for parts replacement or redesign throughout the life of the legacy system.

1.2 Objectives

As discussed in the previous section, the POMT program, begun in 1998, had the following major objectives:

- Extend the CEENSS methodology as a form, fit, function, and interface (F^3I) specification strategy. This involves applying simulatable specifications at each design level, which may potentially require reengineering attention (subsystems, boxes, boards, components and design reuse elements).
- Design products at a higher level by moving from implementation-specific design to abstract level design. This can be demonstrated by designing with behavioral VHDL as opposed to register transfer level (RTL) VHDL, or even designing at a higher level of abstraction, such as SLDL. The benefits of designing at a higher level include:
  - Significantly reduces the reengineering effort
  - Provides a level of component technology independence
  - Improves the potential to resynthesize a component or a board from its simulatable specification
- Partially automate the test development process. The objective was to do the following:
  - Provide CAD tool support for test vector generation
  - Support direct generation of tests from the product requirements specification in the simulatable specification.

1.3 Statement Of The Problem

The Department of Defense (DoD) is faced with a major avionics parts obsolescence problem, which is just beginning to be appreciated. This problem affects not only older legacy avionics systems but also relatively recently developed avionics systems as well as next generation

avionics systems yet to be designed. The root cause of this obsolescence problem is the rapidly increasing rate of electronics parts technology evolution, which is resulting in much shorter life cycles for existing parts. This shortened parts life cycle has resulted in fielded products that have problems acquiring replacement parts soon after their deployment. This situation has been accelerating and is now to the point at which weapon systems have component availability problems before they can even get to production.

This problem has become a major concern for custom component technologies, which depend upon specific processes and facilities such as ASICs, and multi-chip modules (MCMs). When prototype systems are developed early in a weapon system development, there is a good chance that the processes and facilities used for these type of parts may no longer be supported by semiconductor vendors by the time that production begins. This results in the need to reengineer the prototype designs to get to production.

In the case of older legacy weapon systems the parts obsolescence problem is further exacerbated by the fact that the existing design is often not available in a modern electronic representation. In fact, the design often is only available in the form of paper documents, which may or may not be up to date. Furthermore, the parts used in the design often have no functional model, making it difficult to infer the exact function of the part or of the board utilizing the part. This not only makes it difficult to keep these legacy products in operation, but also makes it very difficult to determine what the best strategy is when an obsolescence issue arises.

In the case of modern avionics, there is a shift toward CLS. This will require avionics developers, such as TRW, to warrant their avionics for an extended period. This warranty period may be 20 years or longer. This means that developers must anticipate the effect of parts obsolescence over that period and include in the product cost and system design the impact of mitigating the risk of those parts obsolescence problems. This prospectively means that the developer has to project as many as four or five product reengineering activities for a product over that support period.

A further complication is the fact that during the life cycle of the product, we typically find ourselves in a position in which we must retrofit a product design in order to correct problems, to add or improve functionality and performance, or to take advantage of technology in order to reduce size, weight, power or cost. This implies that when we attack a parts obsolescence situation, we may often find that the reengineering task may not be as simple as providing an exact equivalent capability. Rather, we may need to allow for design changes at the same time.

An analysis of the situation results in the identification of the following key needs:

A) The ability to recognize and predict parts obsolescence problems
B) The ability to identify the most affordable response to the parts obsolescence situation, which may include the development of a drop-in replacement part, the reengineering of the product to utilize a similar part, or the reengineering of the product to utilize alternate component technologies

C) The ability to efficiently recover the designs of legacy electronics, including product requirements as well as implementation
D) The ability to efficiently develop a replacement component
E) The ability to efficiently reengineer an existing design to replace a part
F) The ability to efficiently reengineer to achieve a new design
G) The ability to incorporate new and/or modified functions into a product during the reengineering associated with a parts obsolescence problem
H) The ability to design products from the beginning to minimize the impacts of potential downstream parts obsolescence problems

The focus of this program was upon a reengineering methodology and supporting CAD tools necessary to enable solutions to needs E, F, G and H. The POMT program is not alone in recognizing some of these problems. The E-3 VHDL Synchronizer Program also recognizes similar problems and needs, as stated in their objective:

> *Further complicating the problem are the ever growing diminished manufacturing source (DMS) issues inherent in a system whose basic technology is now over 25 years old. Ever shrinking technology lifecycles which today last perhaps five years or less, reinforce the need for a design methodology which captures the system's functionality in a "technology independent" manner, reducing future impacts of DMS.*

Another area of concern is the lack of support from commercial vendors for DoD applications. As stated in Anthony Bumbalough's white paper "USAF Manufacturing Technology's Initiative on Electronics Parts Obsolescence Management":

> *Commercially manufactured electronics and parts obsolescence issues are intimately intertwined. The military market volume is dwarfed when compared to the consumer market. This has resulted in virtually no military influence on the electronics market place and thus no impact on electronics parts availability. A lot of integrated circuit (IC) manufacturers have stopped producing military-qualified devices. Obsolescence problems have become systemic and chronic in military as well as commercial systems. The Engineering Manufacturing Development (EMD) stage of some weapon systems can take 5 or more years. Production phases are stretched out over several more years and the actual system mission life extends over decades. With some systems, such as the B-52, approaching 100 years. This is in comparison with some key commercial technologies having 18-month product life cycles. Another example of worsening obsolescence problems is commercial consumer electronics is moving to lower voltages (3 volts, 1.8 volts etc.). Current military electronics are based on 5 volt ICs.*

The above discussion points out some key differences between the commercial and DoD markets. With such a small percentage of market share (currently estimated at less than .2%), DoD is unable to drive the marketplace. This not only applies to ICs but also to electronic design automation (EDA) tools as well.

1.4 Terminology

This section identifies and defines some of the terminology used throughout this document.

- Axiom: An *axiom* is a basic truth that is accepted without proof. Axiomatic specification is a technique used where axioms defining pre-conditions and post-conditions for specifications are defined.

- Behavioral: In the discussion that follows, the term *behavioral* will be used to describe designs at higher levels of abstraction than the traditional RTL level. When pertaining to VHDL, it describes VHDL in as non implementation-specific or non technology-dependent manners as possible (i.e., behavioral VHDL).

- Design: The term *design* is used loosely in this paragraph to describe the design of product requirements as well as the design of product structure and the realization of product behavior in algorithms. Design, in its more formal sense, refers to the activity, which occurs in a specific phase of development between the specification of requirements and production from a completed design. Design, as such, is an integrated activity that includes design synthesis, analysis, and validation in a (possibly virtual) prototype.

- Model: The term *model* will describe the expression of that design in an abstract specification or design language.

- Product: The term *product* will describe an object of design at a particular level.

- Reengineering: The term *reengineering* will be used to describe the process of modifying an existing design in order to create a new design.

- VHDL: The acronym *VHDL* will be used to mean IEEE 1076 and any of its extensions. When treating the subject of electronic hardware specification and design methodology, a more explicit aggregation of acronyms would be more cumbersome and less inclusive. When a specific extension, such as VHDL analog-mixed signal (VHDL-AMS), is intended in the discussion, it will be indicated by its more precise designation.

2. APPLICABLE DOCUMENTS

ANSI/IEEE Std 1076-1993: *IEEE Standard VHDL Language Reference Manual,* (1--55937--376--8, IEEE order number SH16840), IEEE, NY, 1993.

ANSI/IEEE Std 1029.1-1991: *IEEE Standard for Waveform and Vector Exchange (WAVES),* (1-55937-195-1, IEEE order number SH15032), IEEE, NY, 1991.

CEENSS Simulatable Specification Report, Contract No. F33615-93-C-4304, April 22, 1997.

CEENSS Methodology Report (Version 3.0), Contract No. F33615-93-C-4304, September, 1997.

BPR and DVTG Methodology Report, Contract No. F33615-98-C-5129, January, 2000.

BPR and DVTG Demonstration Plan, Contract No. F33615-98-C-5129, February, 2000.

VSPEC Language Reference Manual, Version 2.1.1., Information and Telecommunications Technology Center, The University of Kansas, Lawrence, KS, USA. Now available at http://www.ittc.ukans.edu/Projects/SLDG

EDAptive Computing, Inc. Home Page at http://www.edaptive.com

E-3 VHDL Synchronizer Program Final Report, AFRL-ML-WP-TR-1998-4205, December 1998

M. Keating, "*Reuse Methodology Manual For System-On-A-Chip Designs*." Kluwer Academic Publishers, 1998.

D. Barker, "Why Are People Always Talking About VHDL?" AFRL/IFTA white paper.

A. Bumbalough, "USAF Manufacturing Technology's Initiative on Electronics Parts Obsolescence Management" AFRL/MLME white paper

3. METHODOLOGY

The methodology has been divided into two primary sections, one for the design methodology and one for the tools methodology. The design methodology focuses on the CEENSS methodology and appropriate extension for this effort. The tools methodology focuses on the two tools being developed and how they fit into the design methodology.

3.1 Design Methodology

We have identified a goal to make the POMT reengineering methodology and tools support both legacy electronics as well as modern designs. Toward this end we have defined a top-level methodology in which the simulatable specification is the key input to the reengineering process. The POMT methodology is illustrated in Figure 1. The three different processes identified include the following:

- Legacy design
- Initial product design
- Reengineering an existing design.

Each of the design processes has a different starting point, as follows:

- When addressing legacy designs, the methodology begins with the recovery of the existing product design and the representation of that recovered design in the form of a simulatable specification.
- When developing a new design, the first step is the construction of the simulatable specification for the product.
- When reengineering an existing design, the first step is the consideration of the simulatable specification for potential modifications, if necessary.

In all three cases, the process begins with the creation of a simulatable specification and then follows an identical reengineering process flow, as shown in Figure 1.

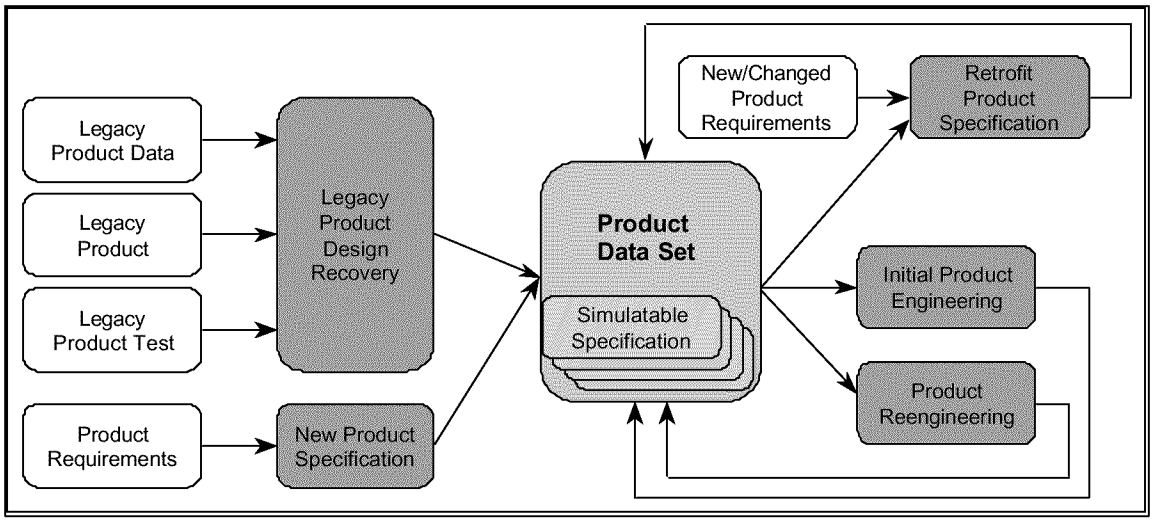

Figure 1. Reengineering Methodology that Supports Both Legacy and Modern Designs

Based upon an analysis of requirements identified by the TRW CNI product development organization in light of parts obsolescence issues and CLS, it was determined that the design and support of modern electronics systems requires that products be designed from the beginning with the expectation that they will have to be periodically redesigned during their life cycle. This is necessary both in response to parts obsolescence issues and the need for product improvements during the life cycle of the product. Further, we have determined that the design and reengineering of products at a higher level of abstraction is a key approach. This allows product design to be accomplished independently of the selected implementation. This directly supports the expectation that products will have to be redesigned, potentially several times. This technique assumes that any individual realization of the product design will incorporate a mapping from a higher level of abstraction to a product implementation. This approach requires an efficient methodology to develop and/or modify an implementation.

It is essential that in addition to designing with the expectation of periodic reengineering, a methodology and supporting CAD tools must be available to significantly reduce the risk and effort required for these periodic preplanned reengineering activities. Two key tool aspects of such a reengineering methodology are illustrated in Figure 2.

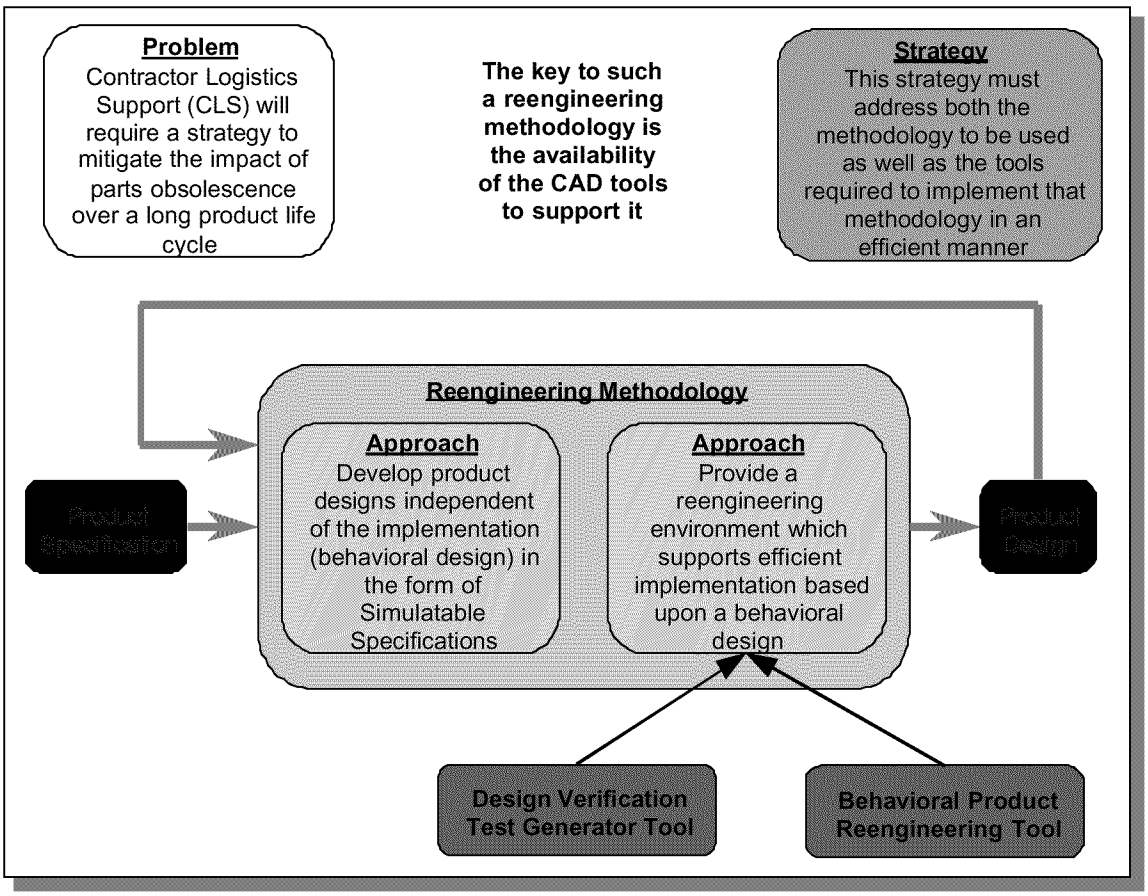

Figure 2. Key Reengineering Methodology Concepts

These two concepts will require CAD tool support in order to make the methodology feasible for application to real avionics product design and reengineering projects.

- We must have an efficient low risk capability to synthesize implementations directly from higher levels of abstraction than traditional RTL VHDL. This is necessary in order to reduce or eliminate the time-consuming and error-prone tasks currently performed to develop the implementation-specific RTL VHDL or gate level designs necessary for individual component synthesis.
- We must have the capability to partially automate the process of developing design verification tests (test benches and accompanying test vectors). This process is currently a highly labor-intensive task as well as being a significant opportunity for both errors and incomplete testing. Typically this is a manual process in which paper requirements specifications are analyzed and formal test vectors/test benches are developed to verify these English language requirements.

The POMT approach was to define such a reengineering methodology and to develop tools to support that methodology. These tools began with the simulatable specification capability developed on the CEENSS program and extended that capability in order to enable rapid reengineering of board and component level designs. This extension consisted of two tool

developments. The first tool development produced a BPR tool which supported the development of product designs directly from the VHDL product behavioral model within the simulatable specification. The second tool development produced a DVTG tool which supported the generation of test vectors directly from the product requirements model within the simulatable specification. These tools provide the following benefits:

- Product design at a behavioral level resulting in designs which are not tied to specific parts or technologies
- Rapid implementation and/or reengineering based upon behavioral synthesis, thus significantly reducing the need for the manual step in which synthesizeable RTL VHDL must be developed and tested
- Rapid design verification following initial design or reengineering. This attacks the highly manual labor intensive test vector generation process.

These tools support a product design/reengineering methodology which concentrates on the specification of products in a simulatable specification form and the rapid mapping of that product specification to an implementation using behavioral synthesis and design verification test automation. This methodology supports the legacy reengineering process as well as the modern product design and reengineering process. If the design is captured in the form of a simulatable specification for legacy systems, then this reengineering methodology and tools will be applicable to legacy reengineering as well.

The following subparagraphs discuss in greater detail the flows that were mentioned briefly earlier in this section:

- Recovering Legacy Designs (no simulatable specification representation)
- Creating a New design
- Reengineering an existing design (existing simulatable specification representation)

There are many flows in the design development process, and even though the identified flows are not intended to be all-inclusive, they do highlight the most common design development flows. Additionally, there is a flow associated with modifying requirements that can be applied to each of the three flows identified above.

3.1.1 Recovering Legacy Designs

When recovering an existing product design, the recovered data may be minimal or may contain a complete product data set. The three sources of information identified in Figure 3 include legacy product data, the legacy product, and the legacy product test. A brief description of each is given in the following paragraphs.

- The legacy product data can come in numerous forms. It can be represented by schematics, hardware description language (HDL) code, source code, initial design requirements, or basically anything else that provides details on the product. It is a representation of all available information on the initial product.

- The legacy product is the actual product. If the original product were a circuit card, the legacy product would be a physical copy of the circuit card. This would obviously be most valuable for characterization purposes if any reverse engineering is necessary to understand the function and requirements of the initial product.
- The legacy product test includes any information used to test the original product. This may include among other things, a test bench with associated test vectors.

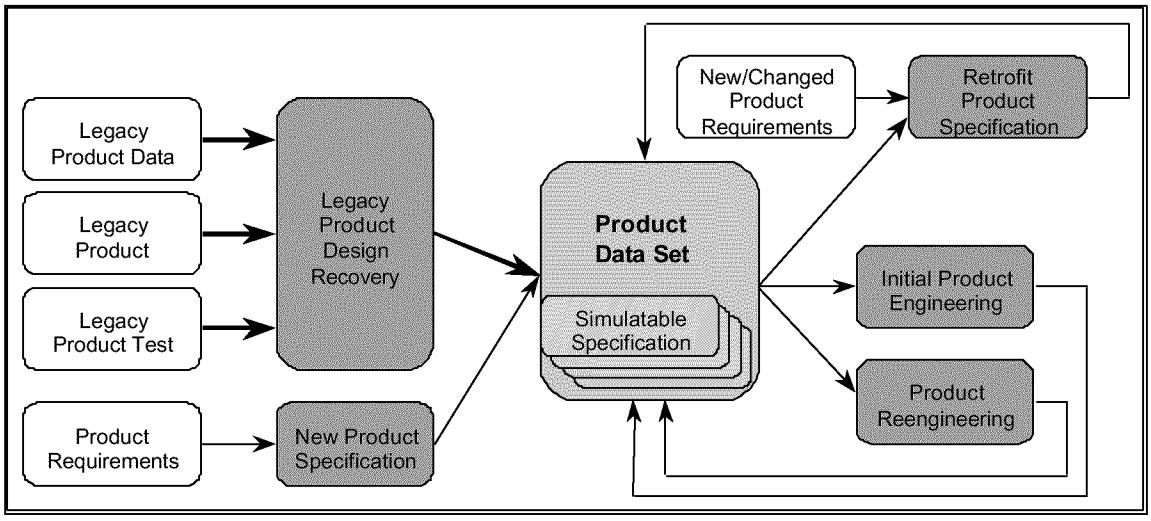

Figure 3. Legacy Reengineering Methodology

The first step in this process flow is to recover and evaluate the existing design as illustrated in Figure 4. A decision must then be made to determine whether or not to implement the simulatable specification methodology or to simply use the classical design environment. It is possible that the legacy product:

- is not intended to have a long life cycle
- does not have any projected reuse components
- can be reengineered within the classical design environment, implementing any required modifications to the legacy design.

If all of the above conditions are true, it may not be worthwhile to implement the simulatable specification methodology. In cases in which it is determined that the simulatable specification methodology would be beneficial, the next step is to recover the requirements and function of the legacy design. Once this is accomplished, a requirements specification for the legacy design is developed. Concurrently, all applicable portions of the legacy design should be brought into the design specification. At this point, the design specification must be evaluated to determine how much of the recovered legacy design data can be applied to the new design. Depending on the completeness of the existing design, the reengineering activity may be viewed as an initial product engineering task or a product reengineering task. Each of these flows will be described in greater detail in sections 3.1.2 and 3.1.3.

There are associated problems with recovery of legacy design. Difficulties include the following:

- Required data does not exist (need to reverse engineer to recover data)
- Existing data is in paper form (not electronic) and converting to electronic form introduces possibility for error
- Electronic data may not be compatible with modern design environments
- Design data is physical design (parts list, board layout, etc.); it does not provide models of design unit
- Difficult to reverse engineer requirements; you are typically reverse engineering an implementation of the requirements
- Few if any tools exist to assist in recovering legacy design.

There are also problems with the current legacy recovery environment, including the following:

- Expensive (consumes significant resources)
- Time consuming (requires reverse engineering for understanding of legacy design)
- Manual and error prone (little or no automation currently available in translation of legacy data into useful format)
- Unavailability of commercial tools, although there are some efforts currently underway that are attempting to address the problem.

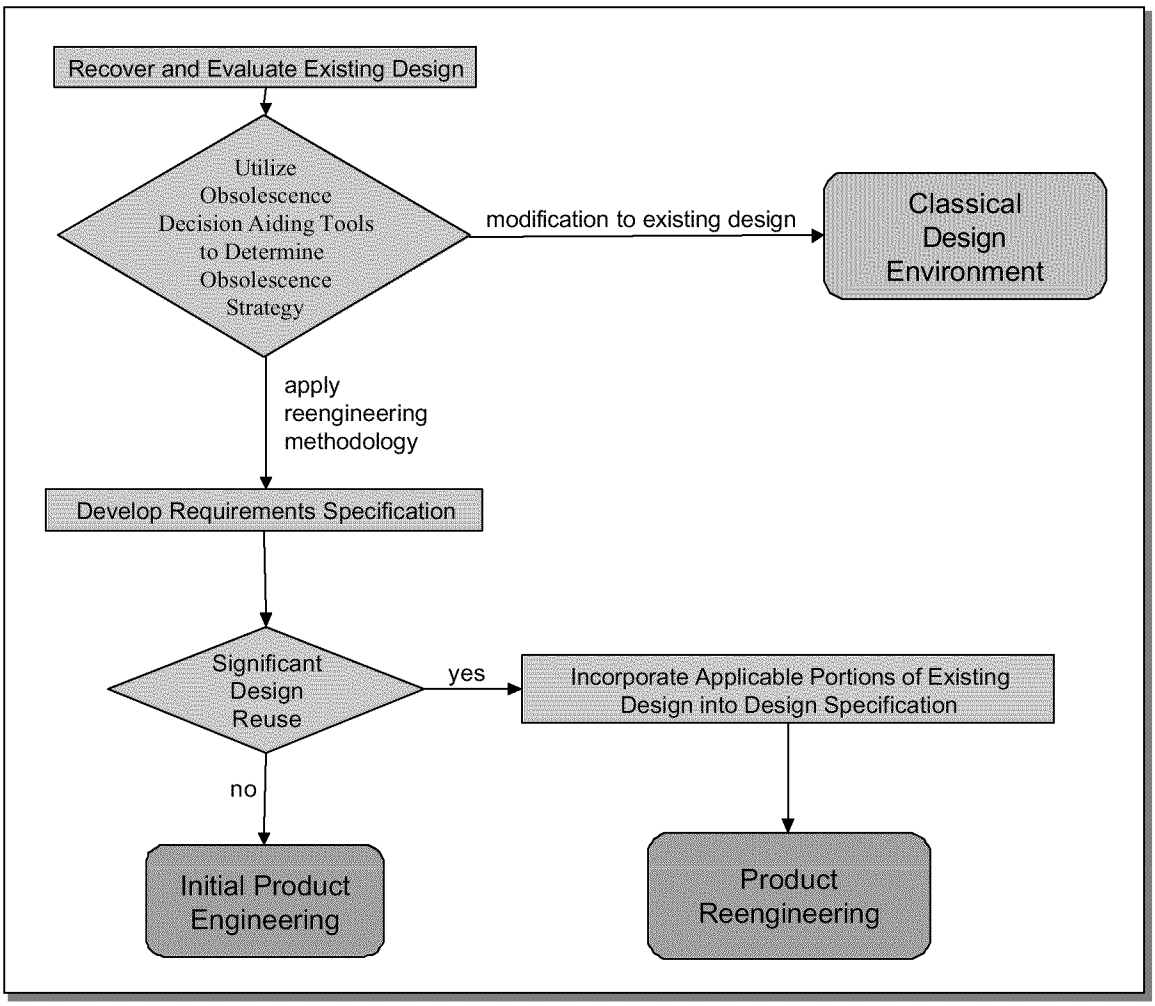

Figure 4. Legacy Recovery Flow

The VP Technologies company is performing another reengineering tool effort for the parts obsolescence initiative. As described in Anthony Bumbalough's white paper:

> *VP is under a three and a half year agreement to look at a number of powerful new technologies: including virtual prototyping, automated model generators, and behavioral abstractors. These methodologies are being brought to bear on the legacy electronics and COTS parts obsolescence/insertion problem. VP will create a "Redesign Advisor" to evaluate the best approaches to an obsolescence redesign problem, including evaluating costs and scheduling of various upgrade options. They will be conducting performance modeling to evaluate capabilities of target system platforms. They will be developing technologies to automate hardware understanding, including: "Explorers" for circuit extraction at chip and board level, multiple thesis resolution through computer-generated experiments, structural board-level virtual prototype creation, and abstraction to simulatable behavioral prototypes. Testing and*

verification of the tools and libraries will be on the Boeing Corporation's legacy design suite. VP's goal is a 4X improvement in upgrade cost and time.

The potential exists that a VP Technologies tool may be used to abstract the behavior of a legacy design, allowing that information be used to support development of a simulatable specification.

3.1.2 Initial Product Design

When creating a new design, the starting point in development of a simulatable specification is the product requirements. This design flow is based on the CEENSS methodology as shown in Figure 5.

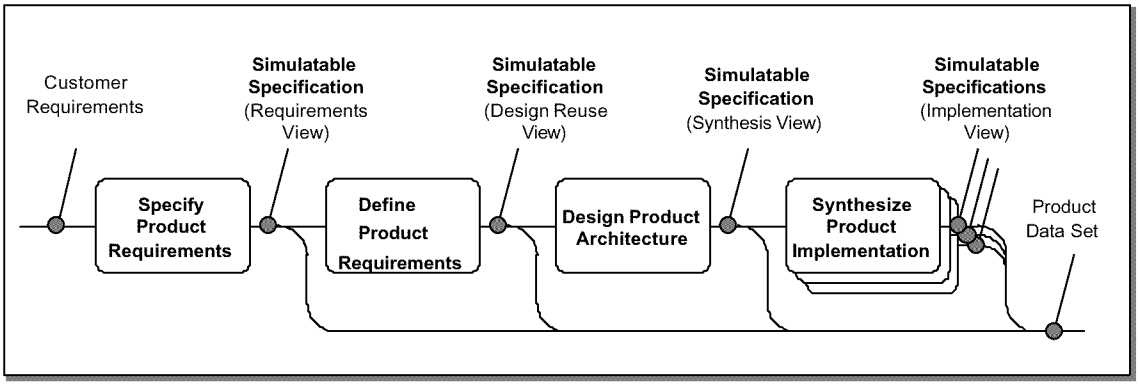

Figure 5. Initial Product Design Flow

The first stage in Figure 5, specify product requirements, refers to the development of the requirements specification. These are the requirements as defined by the customer dictating what the end product needs to accomplish. The second stage, define product requirements, is also associated with requirements, but doesn't deal with the functional requirements associated with specify product requirements, but rather with tying down requirements not dictated in specify product requirements. The next stage, design product architecture, deals with performing the functional decomposition of the design and the corresponding requirements decomposition. Finally, synthesize product implementation deals with the actual design and synthesis of the individual design units. A slightly more detailed breakdown is presented in Figure 6.

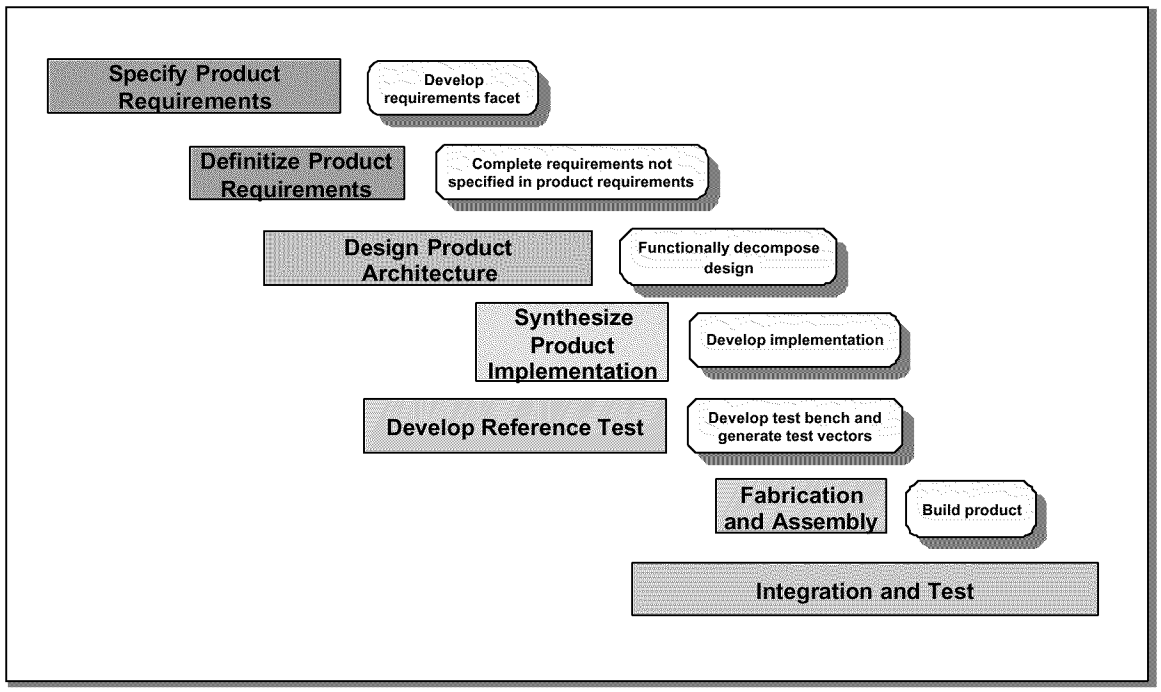

Figure 6. Product Design Flow

Each of the steps in the flow are discussed in the following sections.

3.1.2.1 Specify Product Requirements

The product requirements specification process begins with the customer requirements and results in the creation of the product requirements specification. This process includes the capture and representation of the customer requirements in a structured requirements database, and the development of formal requirements models for the product and appropriate interfaces that are nonstandard or for which an interface model will be developed. At this point, a brief description of the requirements model is probably useful.

A requirements modeling language must be able to express what a module must do without specifying how. For example, what defines in axiomatic terms the required relationship between a function's inputs and outputs. The language must also be capable of specifying nonfunctional performance constraints, such as heat dissipation, electromagnetic interference (EMI) emissions, propagation delay, clock speed, power consumption, form factor, and weight.

The Requirements Model fully specifies the following:

- Required product interfaces
- Required product function and timing
- All *a priori* expectations, in the form of constraints and axioms, regarding operation and attributes of the product.

The requirements model should capture only actual requirements. The actual implementation of the requirements is designed to satisfy the requirements model and describes one valid black box function. The implementation of the design will also be referred to as the operational facet.

Rosetta is the language used to represent the requirements model and is a System Level Description Language (SLDL). Rosetta will be discussed in greater detail in Appendix A.

3.1.2.2 Define Product Requirements

Once the requirements model has been developed, there may be additional requirements that need to be defined in order to proceed with the actual implementation. The step of defining product requirements encompasses completing requirements not specified in product requirements. It involves bringing the requirements up to a point at which they describe the element sufficiently so that it could be a reuse element. This may include the following:

- Completing the definition of all interfaces not tied down in the product requirements
- Completing the timing and electrical constraint definitions where they are not fully defined in the product requirements
- Completing the black box function model in any areas in which the product requirements were not fully functional.

The phrase 'definitize product requirements' was introduced in the CEENSS methodology and is used interchangeably with 'define product requirements' in following figures.

3.1.2.3 Design Product Architecture

Once the requirements have been fully defined, the next step in the flow is to functionally decompose the design. This includes not only the decomposition of the proposed architecture of the design, but additionally the decomposition of the requirements of the design so that requirements flow down to the individual design units. This is a highly iterative design process in which the black box functionality of the product is decomposed to components, and synthesizable simulatable specifications are developed for each unique component. This decomposition should take into account available commercial off-the-shelf (COTS) components, available intellectual property (IP), and business decisions as well as functional and performance requirements. Where the component is COTS, this simulatable specification supports future reengineering of that component. Where the component is custom, this simulatable specification serves as the basis to search for potential IP to use and/or adapt for the current implementation as well as the basis for future reengineering.

3.1.2.4 Synthesize Product Implementation

After the design requirements have been decomposed down to the aforementioned individual design units, the next step is to develop the implementation. This is usually a fairly significant step in the overall process, including the actual design and synthesis of individual design unit.

The synthesizable simulatable specification forms the basis for creating an implementation for each custom component and may utilize IP sub-elements which may or may not be pre-synthesized. The resulting simulatable specification documents the results of the synthesis

activity as well as all of the control and data necessary to replicate this synthesis when necessary. A resynthesis effort based upon the exact same starting point would only need to select a different synthesis target and resynthesize to that target. The BPR tool developed on this effort is focused in this area.

3.1.2.5 Develop Reference Test

A parallel activity to the synthesize product implementation step is the development of the reference test. The reference test development includes the development of the test bench and generation of test vectors and is based upon the requirement model and validates any valid implementation (of which the reference model is one). Validation consists of testing the reference test to determine correct test function and the testing of the reference model using the reference test. The DVTG tool developed on this effort is focused in this area.

3.1.2.6 Fabrication and Assembly

Fabrication and assembly is the development of a physical copy of the design. This is the phase in which the product is actually built. This often involves the development of a physical prototype for use in integration and test before going into full production, but with the increasing acceptance of the virtual prototype, there is also increased confidence in passing over the physical prototype stage. The virtual prototype refers to a model of the design that can be simulated and tested without the actual development of the physical prototype. Increased capability of tools that validate the virtual prototype has led directly to their increased acceptance with the obvious advantage of being able to uncover design flaws before the design is committed to hardware.

3.1.2.7 Integration and Test

Integration and test in previous design methodologies followed the fabrication and assembly step and involved the test and validation of a physical copy of the design. It is now becoming a more parallel process to the synthesize product implementation step with the advent of the virtual prototype. With the ability to simulate a design and exercise the virtual design against the test bench and associated test vectors, the design can be validated before committing to hardware. This does not mean that the integration and test step is completed prior to fabrication and assembly, but rather that there is increased confidence in the design before physical development, leading to a decreased possibility of uncovering errors after commitment to hardware.

3.1.3 Reengineering an Existing Design

There are many different scenarios when considering the reengineering of an existing design. As shown in Figure 7, the primary reengineering scenarios include the following:

- Redesign (system level requirements reengineering)
- Board level reengineering (system level architecture reengineering)
- Component level reengineering.

Depending on what change is causing the reengineering activity determines at what point in the design process the redesign activity needs to start. More specifically, if there was a fundamental requirements change that affects the entire system, the redesign activity would need to revisit product requirements. Effectively, this means the redesign activity needs to start from almost the beginning. On the other hand, if the change is at the component level and does not change the component's requirements, the redesign activity can be isolated to that component, thereby limiting how far back in the design cycle to implement the change. The following sections will examine each of the possible scenarios.

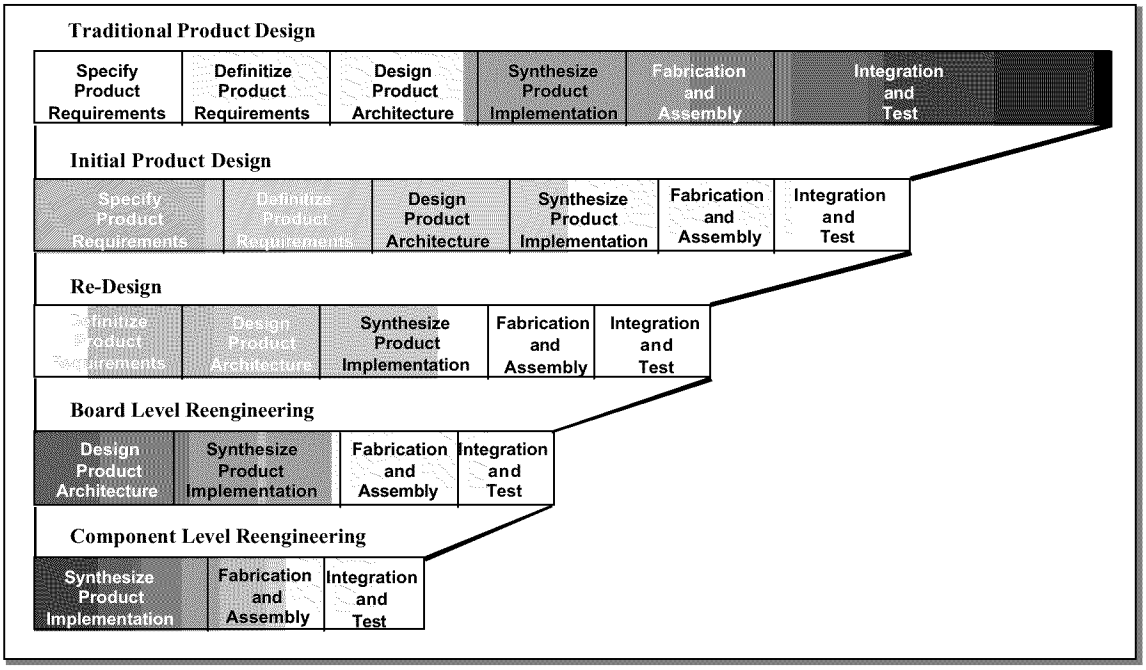

Figure 7. Engineering and Reengineering Flows

3.1.3.1 Redesign (Modifying Requirements)

When discussing requirements for a design, there are effectively two types of requirements to be considered. The first are the product's requirements as defined by the customer. These requirements represent what the product needs to do to meet the customer's definition of correctness. The second type of requirements is associated with what needs to be defined in order to proceed with the design and implementation. These have also been referred to as implementation requirements and are often associated with activities such as defining the interfaces to the product. These were discussed briefly in sections 3.1.2.1 and 3.1.2.2. Although both of these activities are associated with the product requirements, there is a need to be able to differentiate between the two in the interest of reengineering. Figure 8 and Figure 9 portray the different entry points into requirements modification, depending on whether a customer requirement or implementation requirement change is forcing the reengineering activity.

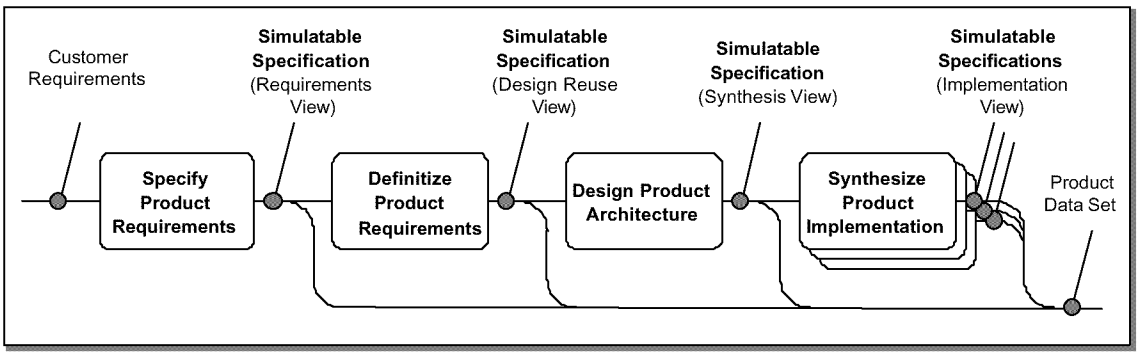

Figure 8. Modifying Product Requirements

In the case of a modified customer requirement (modified product requirement), the entire specification needs to be revisited, whereas in the case of a modified implementation requirement (modified product definition), the possibility exists that the resulting impact may not even affect the reference test. This highlights the importance of differentiating between the types of requirements along with the difficulty associated with recovering a legacy design. When evaluating a legacy design, it is often next to impossible to determine whether a specific function was dictated by the product requirements or was simply the result of a design decision associated with the implementation.

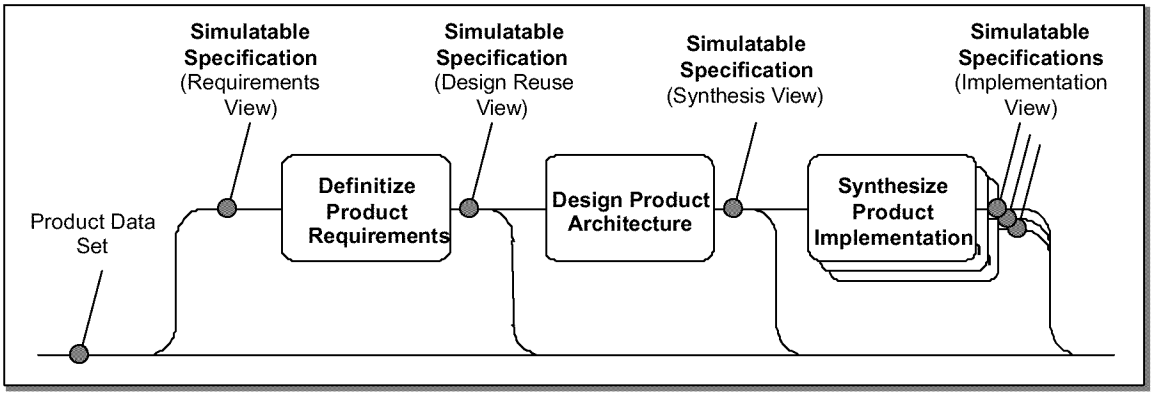

Figure 9. Modifying Product Definition

Figure 10 is a slightly modified version of Figure 6 and points out that the reengineering activity is very similar to an initial design except that it is modifying product requirements as opposed to specifying the product requirements. The entry point into the design flow in Figure 10 will be either the first or second step depending on what type of requirements change is driving the redesign. For a more detailed description of each of the design steps, please refer to the appropriate subsection of 3.1.2.

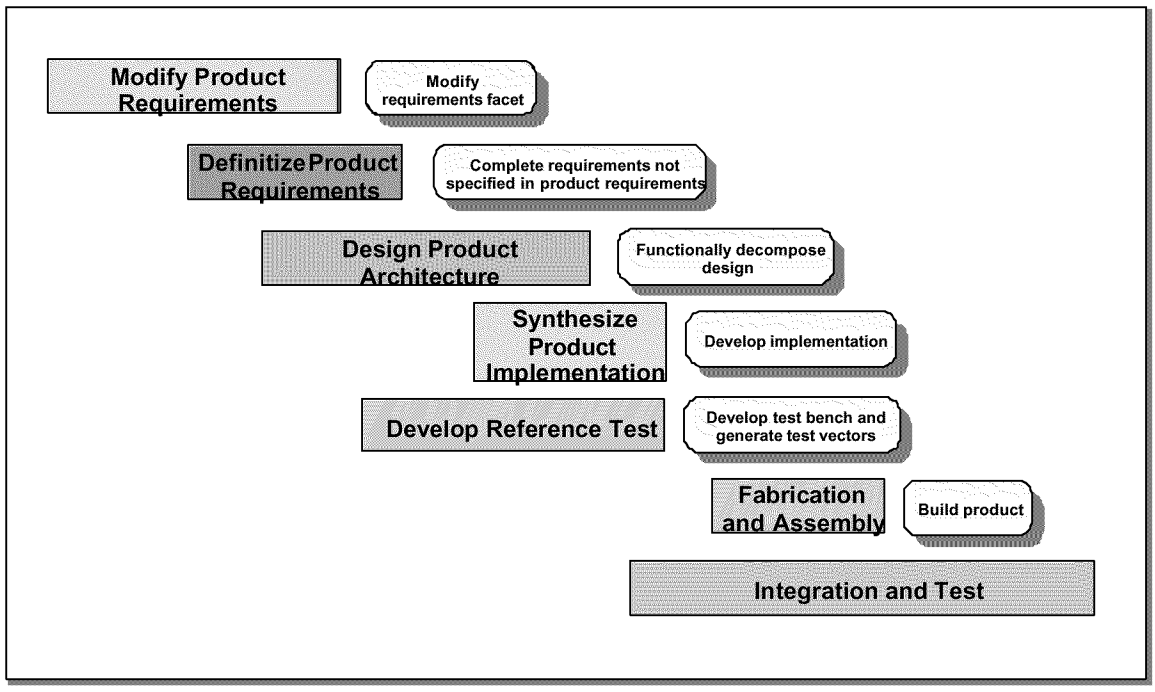

Figure 10. Product Design Flow from Modified Requirements

3.1.3.2 Board Level Reengineering

In the case of a board level reengineering activity, the product's requirements are not changed, but the architecture of the design most likely has. A typical scenario for this type of redesign activity is the case of combining multiple boards into one board or multiple components into a single FPGA or ASIC. The requirements of the product have not changed, but the implementation has. Figure 11 and Figure 12 represent similar views of this particular design flow.

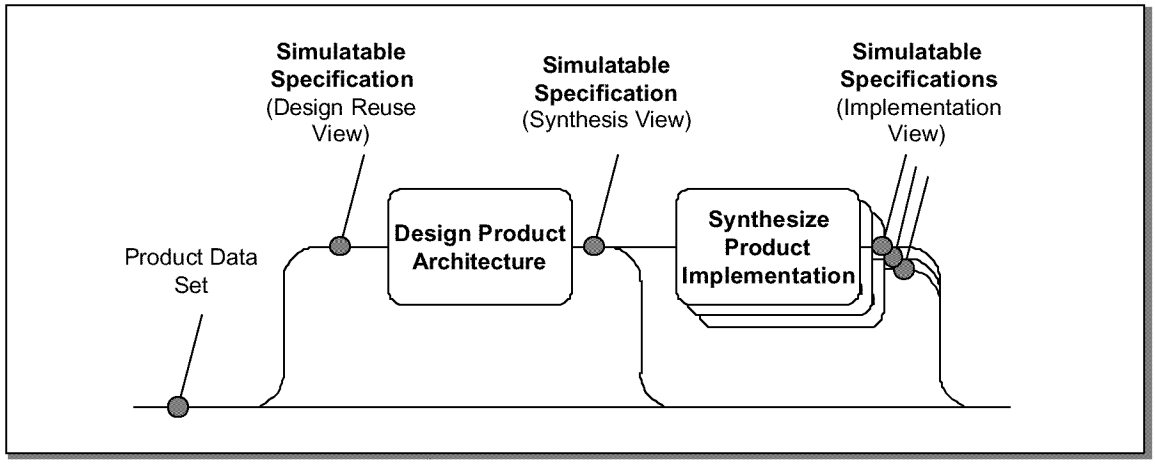

Figure 11. Modifying Product Architecture

In Figure 12, the entry point into the design flow is at the design product architecture step, bypassing both of the requirements steps. Again, it is worth noting that the design activity

follows the same path as the previously described flows, but with a different entry point. For a more detailed description of each of the design steps, please refer to the appropriate subsection of 3.1.2.

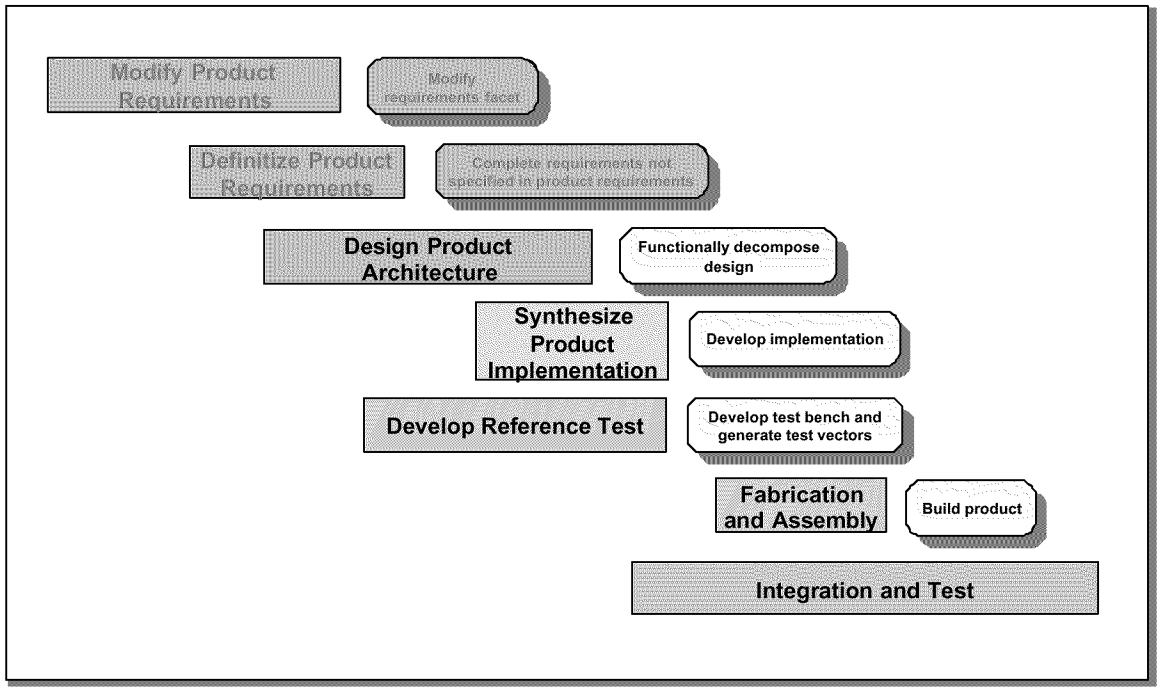

Figure 12. Product Design Flow from Modified Architecture

3.1.3.3 Component Level Reengineering

In the case of a component level reengineering activity, neither requirements nor architecture has changed. This is often merely a resynthesis activity driven by unavailability of a particular component or technology. Figure 13 and Figure 14 represent different views of this particular design flow.

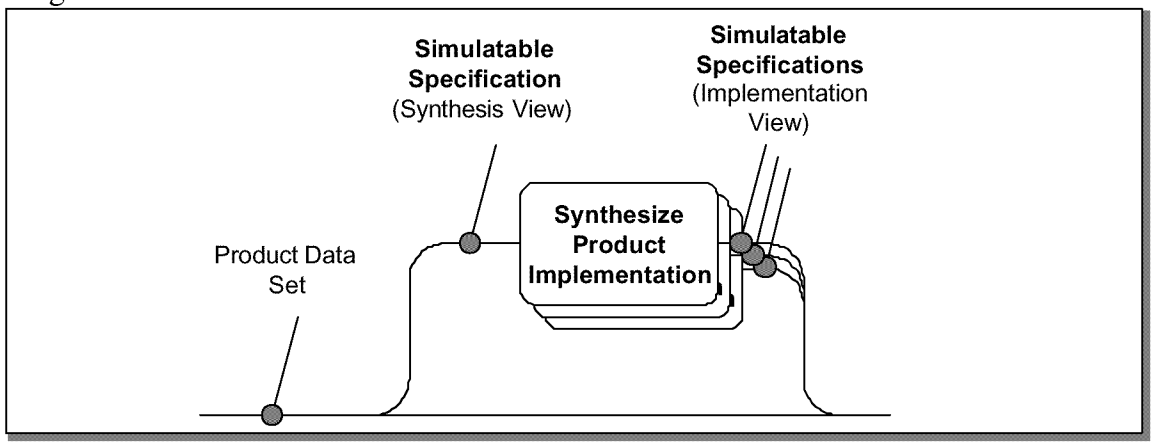

Figure 13. Modifying Product Implementation

In the case of previously described redesign activities, the possibility existed that the test bench would not need to change. In this particular flow however, that is the rule. None of the requirements or external interfaces should be affected by the redesign activity; therefore, the same reference test along with its associated test vectors should still be applicable. For a more detailed description of each of the design steps, please refer to the appropriate subsection of 3.1.2.

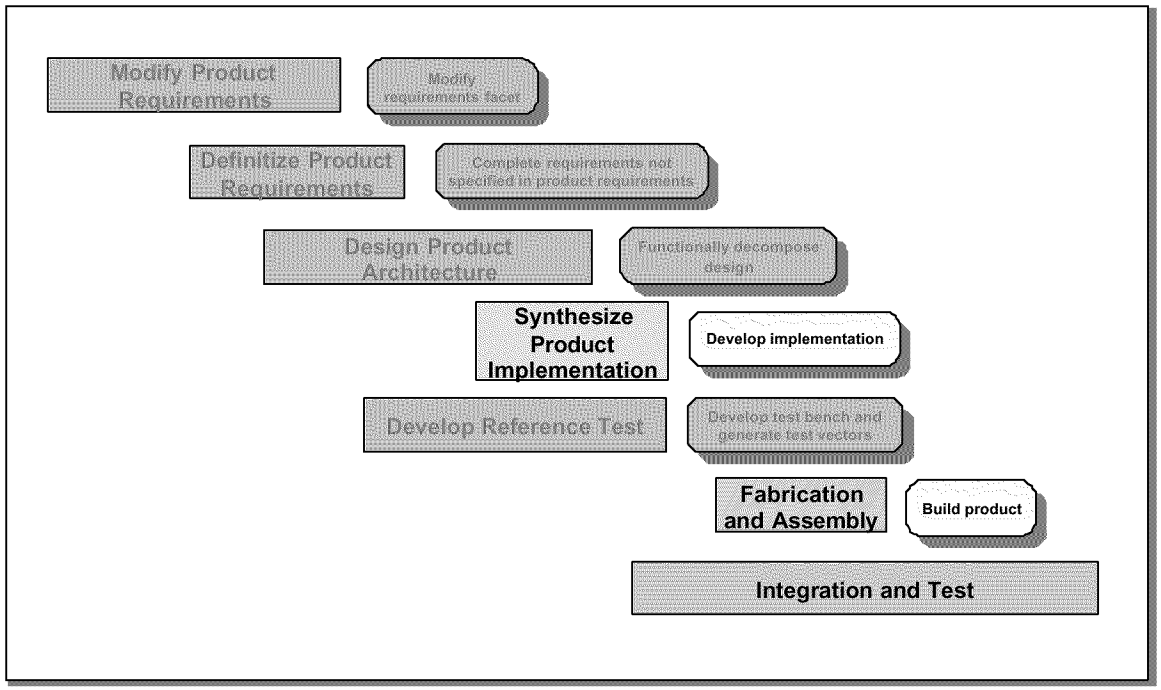

Figure 14. Product Design Flow from Modified Implementation

3.1.4 POMT Design Methodology Metrics

In the discussion of the methodology so far, various design possibilities have been discussed, including both engineering and reengineering scenarios. The design possibilities introduced so far include the following:

- Initial product design (POMT)
- Requirements level reengineering
- Board level reengineering
- Component level engineering.

Additionally, there are two more design methodologies that need to be introduced. These are as follows:

- Traditional product design
- Initial product design (CEENSS).

The traditional product design is referred to briefly in Figure 7 and refers to a common approach to product design. The CEENSS initial product design employs the CEENSS design methodology, which was the starting point for the POMT design methodology.

The following sections discuss various aspects of the individual scenarios as well as provide metrics for the projected benefits.

3.1.4.1 Engineering Scenarios

In the scenarios listed above, three scenarios are associated with initial product design and three scenarios are associated with reengineering. This section concentrates on the initial product design scenarios, which include the following:

- Traditional product design
- Initial product design (CEENSS)
- Initial product design (POMT).

The steps in each of the three design flows are basically the same as illustrated in Figure 6. These include:

- Specify product requirements
- Define product requirements
- Design product architecture
- Synthesize product implementation
- Develop reference test
- Fabricate and assemble
- Integrate and test.

The primary difference between the traditional product design and the CEENSS product design is that in the CEENSS flow, more time is spent on the up-front requirements definition task which has a payoff in later tasks. The most notable improvement was in the integration and test step as demonstrated on the CEENSS program.

The POMT initial product design flow is effectively the same as the CEENSS initial product design flow, except for projected improvements in the synthesize product implementation and develop reference test steps due to the incorporation of the BPR and DVTG tools respectively.

3.1.4.2 Engineering Projected Benefits

Although each of these processes has been described in some detail, no details regarding the projected benefits from a metrics point of view have been discussed. This section addresses these projected benefits.

The numbers represented in this section were gathered from numerous sources, including the following:

- Design experiences of the POMT team
- Interviews with design engineers and engineering managers within TRW

- Various publications.

These numbers are meant to reflect projected benefits of implementing the POMT methodology as well as actual metrics from previous design experience. These numbers are subjective in nature because there are almost as many different design methodologies as there are designs.

The first step in setting up a baseline for metrics development is to determine what percentage of the design flow is spent in each of the individual steps. The traditional product design methodology will be used as the baseline

Table 1 reflects the level of effort of each of these steps for the traditional product design methodology and the CEENSS product design methodology.

Table 1. Traditional Product Design Effort

Design Step	Traditional	CEENSS
Specify product requirements	5	10
Define product requirements	2.5	2
Design product architecture	2.5	2
Synthesize product implementation	15	10
Develop reference test	20	14
Fabricate and assemble	5	5
Integrate and test	50	25
Total	**100%**	**68%**

As noted earlier, the CEENSS methodology requires additional effort in the specify product requirements step but offers improvement in most of the other steps. Most notably, there is a significant reduction in the integration and test step due to the increased effort spent in requirements. It is obviously more beneficial to capture an error in the requirements stage as opposed to integration and test.

The POMT methodology is effectively implementing the CEENSS methodology along with the tools being developed on POMT. The tools will be discussed in detail in section 3.2. For the purposes of this metrics discussion, all that is necessary to understand about these tools is that they are focused in the synthesize product implementation and develop reference test steps. The projected benefits of this methodology, along with the previously described methodologies, are shown in Table 2.

Table 2. Projected Benefits of POMT Methodology

Design Step	Traditional	CEENSS	POMT
Specify product requirements	5	10	10
Define product requirements	2.5	2	2
Design product architecture	2.5	2	2
Synthesize product implementation	15	10	5
Develop reference test	20	14	7
Fabricate and assemble	5	5	5
Integrate and test	50	25	25
Total	**100%**	**68%**	**56%**

3.1.4.3 Reengineering Scenarios

There are three scenarios associated with product reengineering. These include the following:

- Requirements level reengineering
- Board level reengineering
- Component level reengineering.

Each of these scenarios is based on having a simulatable specification as the starting point. In order to provide metrics to project the benefits of each of these reengineering flows, a baseline needs to be introduced depicting the current approach to reengineering.

In the *Reuse Methodology Manual for System-On-A-Chip Designs*, Michael Keating and Pierre Bricaud describe the difficulty with the current reengineering scenario:

> *Legacy designs – those designs we wish to reuse but were not designed for reuse – present major challenges to the design team. Often these designs are gate-level netlists with little or no documentation. The detailed approach to a specific design depends on the state of design. However, there are a few general guidelines that are useful. …*
>
> *The most difficult part of dealing with a legacy design is recapturing the design intent. With a good functional specification and a good test suite, it is possible to fix, modify, or redesign a block relatively quickly. The specification and test suite fully define the intent of the design and give objective criteria for when the design is functioning correctly.*
>
> *If the specification and test suite are not available, then the first step in reusing the design must be to recreate them. Otherwise, it is not possible to modify the design any way, and some modification is nearly always required to port the design to a new process or to a new application.*

The problem with recreating the specification and test suite, of course, is that these activities represent well over half of the initial design effort. Almost none of the benefits of reuse are realized.

Thus, if the specification and test suite exist and are of high quality, then reuse is easy, in the sense that even if a complete redesign is required, it will take a fraction of the cost and time of the original development. If the specification and test suite do not exist, then reuse of the design is essentially equivalent to a complete redesign. …

In spite of the observations in the above section, some unfortunate design teams are required to try to reuse existing designs, usually in the form of netlists, for which documentation and test benches are mostly nonexistent. In such cases, most teams attempt to use the design as-is. That is, they attempt to port the design to a new process without changing the functionality of the circuit in any way.

Formal verification is particularly useful in this scenario because it can prove whether or not modifications to the circuit affect behavior. Thus, synthesis can be used to remap and reoptimize the design for a new technology, and formal verification can be used to verify the correctness of the results.

The following are several key points discussed in the above excerpt:

1. It is difficult to extract design intent when reverse engineering a product design.
2. Depending on the information available, reverse engineering the product can be anywhere from extremely difficult (if not impossible) to relatively straightforward.
3. If the product was not designed originally with design reuse as a goal, reengineering the product can often result in a complete redesign.
4. A significant portion of the reengineering activity is involved in extracting the requirements of the original design.

For the purposes of this baseline, it was assumed that the original product was not designed with reuse in mind. This will lead to an almost complete redesign activity when reengineering the product, which is essentially equivalent to the traditional initial product design. The details of the primary differences between the other reengineering flows were described in detail in section 3.1.3.

3.1.4.4 Reengineering's Projected Benefits

What remains to be discussed are the benefits of the different reengineering methodologies. One of the greatest benefits with the POMT (or CEENSS) methodologies is that they help to minimize the possibility of starting from scratch when faced with reengineering the product. Another benefit, as described earlier, is that the driving force for the redesign helps dictate that

step in the design methodology that the redesign activity needs to start from. This was described in detail in section 3.1.3. Table 3 portrays the projected benefits for each of these design scenarios.

Table 3. Reengineering Design Effort

Design Step	Traditional	Requirements Level	Board Level	Component Level
Specify product requirements	5	5	0	0
Define product requirements	2.5	1	0	0
Design product architecture	2.5	2	2	0
Synthesize product implementation	15	5	5	5
Develop reference test	20	3.5	0	0
Fabricate and assemble	5	5	5	5
Integrate and test	50	25	20	15
Total	**100%**	**46.5%**	**32%**	**25%**

When evaluating the metrics for the reengineering activities, it is important to note that some of the steps have zero effort because these steps should not be necessary within the particular redesign activity.

3.2 Tools Methodology

Two goals of the POMT effort were to create a methodology that supports the engineering/reengineering process and to develop tools that can be used within that process. The focus to this point has been on the methodology itself; discussion now shifts to the tools. More specifically, the goal of the tools is to provide an engineering/reengineering environment that supports efficient implementation based upon a behavioral design. There are many tools currently available that support the reengineering methodology, but there are also some identified gaps in the tool process that will be the focus of the POMT effort.

As described by Anthony Bumbalough in his white paper:

> *The goal of this area is to provide the military industrial base and logistics centers with common, commercially available obsolescence management … reengineering tools. The intent is for the tools to be maintained and supported by vendors who also have both commercial and military customer bases. While the data input tools and the resulting designs may be different, it is important that both commercial and military industry use the same tools and standards for information exchange.*

More specifically, relative to the POMT objectives described in section 1.2, two of the primary objectives were as follows:

- Design our products at a higher level by moving from implementation-specific level design to abstract level design. This can be demonstrated by designing with behavioral VHDL as opposed to RTL VHDL, or even designing at a higher level of abstraction such as SLDL. The benefits of this include the following:
  - Significant reduction of the reengineering effort
  - Provision of a level of component technology independence
  - Improvement in the potential to resynthesize a component or a board from its simulatable specification
- Partially automate the test development process. The objective was to do the following:
  - Provide CAD tool support for test vector generation
  - Support direct generation of tests from the product requirements specification in the simulatable specification.

The first objective was the focus of the BPR tool, developed by Synopsys. The second objective was the focus of the DVTG tool, developed by the University of Cincinnati and commercialized by EDAptive. Each of these are discussed in greater detail in following sections.

The term simulatable specification has been used extensively throughout this report and is a key component of POMT's methodology. A part of the simulatable specification that has also been receiving considerable attention is the requirements model. At this point, it may be beneficial to describe what is meant by the term requirements model and what its benefits are. Darrell Barker's white paper describes requirements modeling as follows:

> *Requirements modeling … models a system's requirements. Another way to say this is that it is a "meta-specification"; which means a specification of a specification. The research is based on the fact that studies (e.g., the Standish Report, 1995) have determined that 5 of 8 leading causes of cost/schedule overruns, project failures, and poor quality are related to the system's requirements and specifications. This is one of the reasons that VHDL behavioral modeling is so effective. But the requirements modeling referred to here is done prior to VHDL behavioral modeling. Requirements modeling models the intended functionality of the system, the control of the system, the structure of the system, the constraints on the system, the information content of the system, and the verifications that must be performed on the system. Requirements modeling does not model an implementation of the system; VHDL does that. Requirements models enable computer analysis and tracking of requirements. This will help to manage large system developments, to control specialists like programmers and IC designers, and to get many more designs right the first time. Requirements models will enable automatic parameter passing to synthesis tools, automatic system structure passing to Electronic Computer Aided Engineering and Computer Aided Software Engineering tools and to cost and failure analysis tools. Requirements models are the very first models for a system and*

provide an early kind of prototype to show the customers as well as enable system architecture trade-off. They will enable automatic searching for re-usable designs. They will enable automatic verification and validation. And they will enable intended functionality to be passed to "System Synthesis" tools. Requirements models are implementation independent. Therefore the requirements can be met with any combination of hardware, software, analog, digital, or microelectromechanical implementation. Requirements modeling will reduce system costs and schedules of projects that do not have major problems from 20-40% and will cut down on the frequency and severity of cost/schedule overruns by eliminating many of the surprises that are only discovered during systems integration. Requirements models will provide the necessary information to maintain, upgrade, or re-engineer a system. Requirements modeling is a new dimension and direction in the modeling and simulation spectrum. Requirements modeling uses a new language called Systems Level Design Language.

The methodology for each of the tools are addressed next.

3.2.1 BPR Methodology

The BPR tool provides a design environment where the product engineer can develop a product implementation based upon the behavioral product model contained within the simulatable specification. This tool supports the rapid development and modification of design implementations, taking advantage of commercial behavioral synthesis tools currently available.

3.2.1.1 Goals and Objectives

The BPR tool is comprised of commercially available tools (both shrink-wrapped and customized tools) and has been developed by Synopsys to support design reuse, design reengineering, and parts obsolescence management. Synopsys has utilized its expertise in this area to develop the appropriate tools, flow, and overall environment to allow design re-targeting and reengineering with a wide variety of targeted technologies.

The overall purpose of the POMT program is to reduce the engineering effort and cost associated with reengineering legacy FPGA and ASIC designs. Currently, such legacy designs outlive the process and manufacturing technology for which they were originally produced. Thus, to ensure the continued delivery of critical components, redesign of such circuits in the current technology is required. Furthermore, in order to realize the cost and overall system complexity reduction benefits associated with the higher levels of integration offered by new technologies, the reengineering effort is compounded.

Much of the difficulty and cost associated with current reengineering and obsolescence management is a result of the implementation-specific details imposed upon the design during its original realization. With the passage of time and advent of new technologies, such implementation-specific details may not be applicable for the current design processes, manufacturing technology, or targeted level of integration, and thus represent an entangled web

that the design team must sort out prior to reusing or re-implementing the targeted design. It is thus required that an environment and flow be established that frees the representation of the design from the implementation-specific details while enabling the physical realization of the design at hand. Thus the technology and intellectual property represented by the design can be easily re-targeted to new silicon process technologies and/or integrated into a larger design at a future date with relative ease.

In order to support the changes in design processes that may occur over the life of the topic legacy designs, an obsolescence management system must be able to accept design representations of varied levels of abstraction. Currently, this includes anything from behavioral VHDL down to a gate level netlist. With the advent of SLDL, the level of abstraction is raised even further. Furthermore, the design flow, including all inputs, variable settings and constraints for each step, used to achieve the final representation must be clearly identified. For this flow to be reproducible, either as-is or as migrated to the then current best practice, it is critical that industry standard tools and data formats be used. To facilitate the adoption of new and current best-practice design flows, the obsolescence management system must be open and flexible enough to enable integration of new tools and flows as they emerge.

Given the rapid change and feature size reduction of silicon process technology, a reengineering flow must also be able to easily accept current silicon vendor libraries. In addition, the tools and the associated outputs of the reengineering flow must be widely accepted across the myriad of silicon vendor design flows.

To enable higher levels of integration and the associated benefits mentioned above, an obsolescence management system must clearly capture and document the items referenced previously and provide an environment in which they can be reproduced and integrated into the larger design. Such documentation entails far more than a textual description. Rather, it is the organized collection and description of all the necessary inputs and processes required to realize the subject module. The obsolescence system must not only clearly capture such information, but also must enable its reuse and integration into the larger design.

Section 3.2.1.2 describes how the BPR tool meets the above objectives and provides an obsolescence management methodology whereby an implementation-independent representation of a design is taken as input, architectural trade-offs explored, and the implementation-specific outputs generated to enable a physical realization of the design.

3.2.1.2 Implementation

3.2.1.2.1 Technical Approach

Figure 15 shows a top-level tool flow that incorporates the BPR tool into the design environment. Figure 16 provides a more detailed view, showing the relationship of existing technology and tools currently provided by Synopsys for the POMT design flow. This solution

leverages commercial products with additional development to specifically support the design object representation required by an affective parts obsolescence management environment.

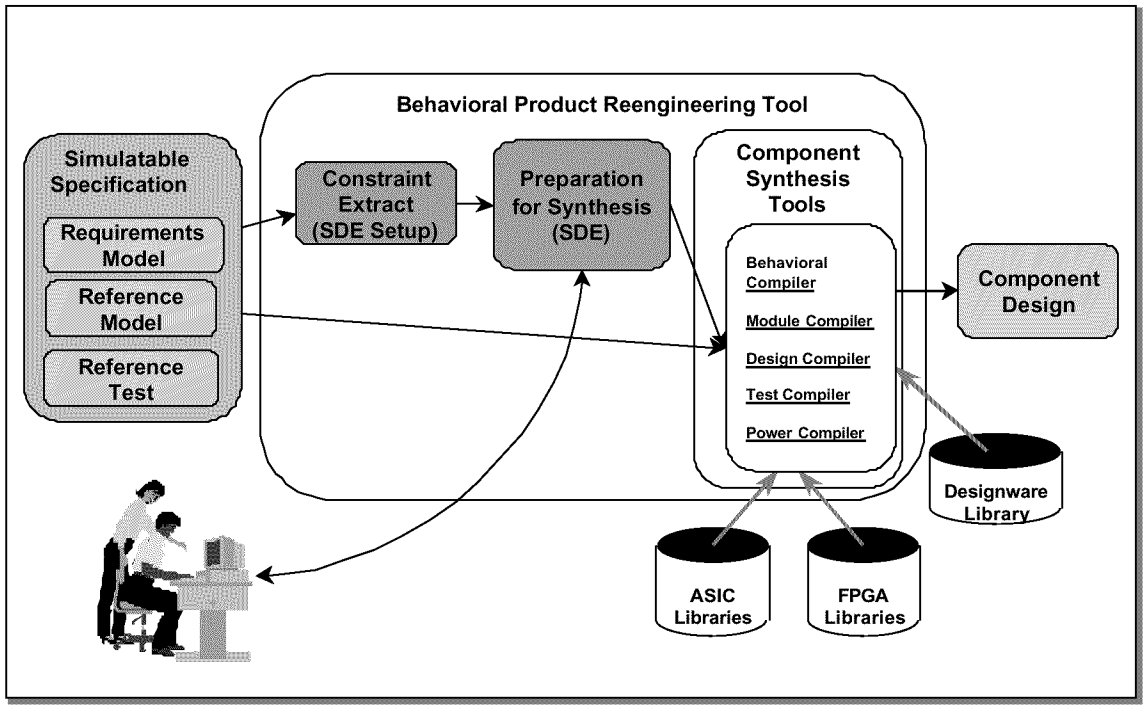

Figure 15. BPR Tool Flow

The approach developed by Synopsys is built around the SDE tool, developed by the Synopsys Professional Services Group, along with tools that are in commercial release. The use of COTS tools for cost savings, as well as capability, was one of the goals of this approach. Key development efforts for the purpose of the POMT demonstration flow included the integration of Synopsys' FPGA Compiler tool into the SDE environment and the development of the SimSpec2SDESetup tool to automate the flow through of information from the simulatable specification into SDE and the physical realization of the design.

The simulatable specification is the implementation-independent representation of the design data associated with the topic design. The simulatable specification contains such information as source file level of abstraction, file location, and targeted tool usage, as well as top-level constraints intended to represent the environment in which the design will be placed. The SDE then serves as the means to translate this implementation-independent information into a design format that is physically realizable. Hereafter this flow will be referred to as the SimSpec/SDE flow.

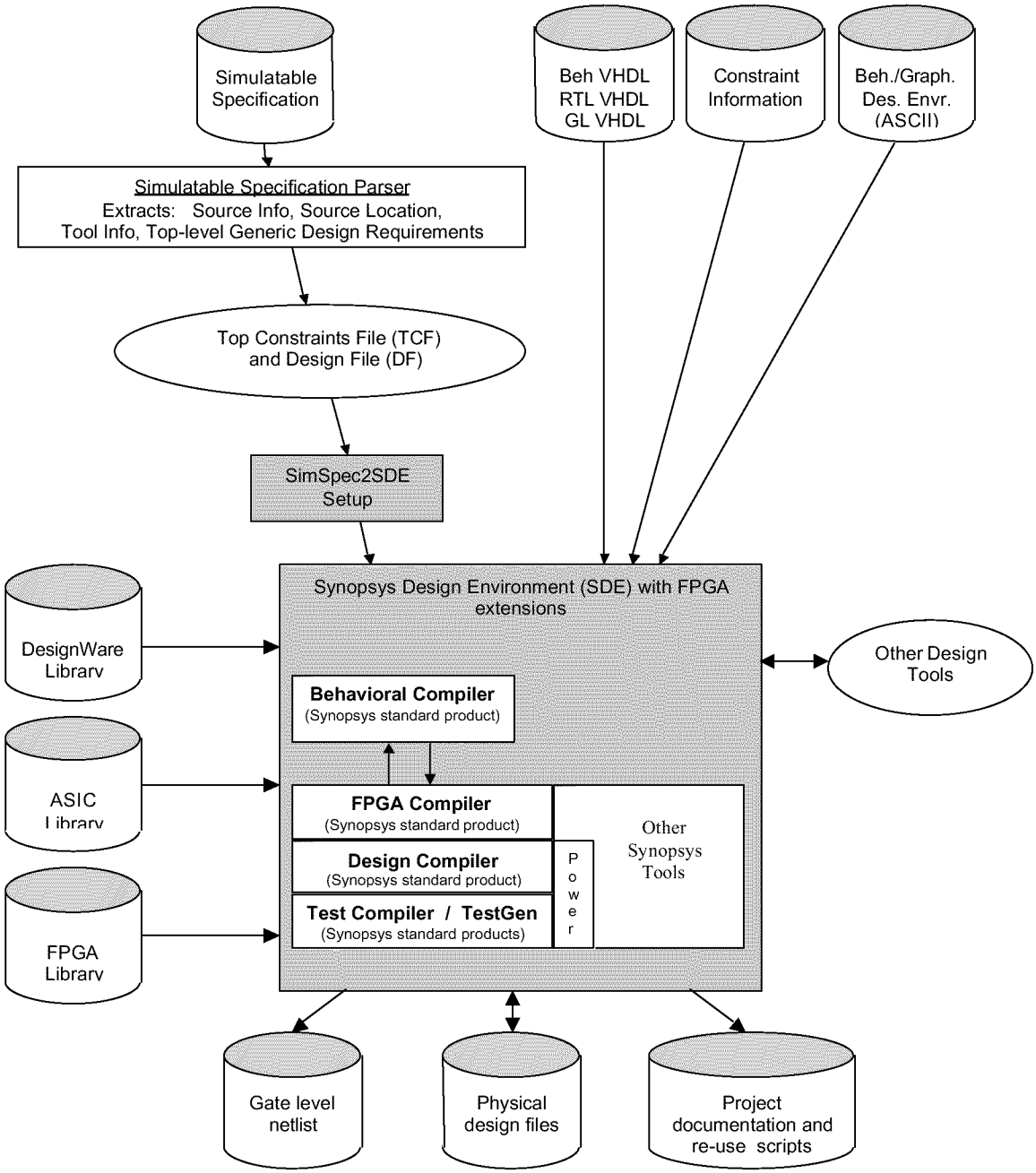

Figure 16. SimSpec/SDE Design Flow

3.2.1.2.2 SimSpec/SDE Design Flow Description

Design object data extracted from existing designs or captured from specification and requirements documents are contained in a simulatable specification data set, or as external (stand-alone) files. This data set supports the VHDL language as a description language for the behavior of a design object. The simulatable specification has the ability to reference other

formats of design representation, such as graphical front-ends to design capture (i.e., Mentor Graphics' Renoir). The SDE supports the ability to receive this behavioral or graphical representation after the design data has been written out in compatible American National Standard Code for Information Interchange (ASCII) VHDL format. In addition, design data described in stand-alone files is also supported.

Design data representations embedded or referenced within a simulatable specification data set are extracted through a simulatable specification parser (provided by the DVTG effort) into a design file (DF) and top constraints file (TCF), respectively. As an alternative, these files may also be created by hand. An example design file and top constraints file can be found in Figure 17 and Figure 18, respectively. Note that among other things, the design file contains such information as targeted silicon vendor library, identification of design source and test bench files, location pointers, and target tool information. The top constraints file contains top-level generic design requirements.

The SimSpec2SDESetup tool, developed by Synopsys, reads the design file and top constraints file, and seeds the associated information into SDE, thus ensuring the flow-through of critical top-level requirements. As an alternative, SDE offers an interactive tool to guide the user through the process of supplying all required information. Once this information has been obtained, SDE will translate it as needed into the various control and constraint files required to properly drive the design tools within SDE.

The SimSpec/SDE flow supports behavioral, RTL and gate level (GL) representations of the design. Although the requirements may be represented in Rosetta (SLDL), there is currently no capability to synthesize Rosetta; therefore, the design implementation must be in a form that is synthesizable (i.e., VHDL). The SDE can receive as inputs any combination of input representations and then preprocess the information as necessary for interfacing with the appropriate synthesis and design tools. In addition, SDE provides the user access to basic scripts used to drive design tools. Therefore, these scripts may alternatively be modified for interfacing with non-Synopsys-supported design tools, thus enabling the user to integrate other tools of their choice.

#[Begin FILESPEC]

 # Project Name Section: A project name identifier can be placed here for reference

Project: example # <project name>

 # Top Level of Hierarchy Section: top level of IC design to be implemented

Top_level: Top # <top level name>

This is the level were the design constraints specified in the simulatable specification are applied

 # Top Level of Test Bench Section:

Test_Bench top_tb #<test bench name for top level>

 # Target library Section: identifies the library the IC will be implemented in

Library: /libraries/cba.db abc.db # <full path to library>

 # Source/VHDL file Section: source locations, block names, type, tool to use and whether it

 # is ready for synthesis

# location	**name**	**source_type**	**target_tool**	**synthesizable**
spec0/vhdl/top_tb	top_tb	rtl	none	N
spec0/vhdl/top.vhd	Top	rtl	DC	Y
spec0/vhdl/blockA.vhd	blockA	rtl	DC	Y
spec0/vhdl/blockA_tb.vhd	blockA_tb	rtl	DC	Y
spec0/vhdl/blockB.vhd	blockB	rtl	DC	N
spec0/vhdl/blockC.vhd	blockC	rtl	DC	N
spec0/vhdl/blockD.vhd	blockD	behavioral	BC	Y
spec0/vhdl/block1.vhd	blockA	behavioral	BC	Y
spec0/vhdl/block2.vhd	blockA	rtl	DC	Y
spec0/vhdl/block3.vhd	blockA	rtl	DC	Y
spec0/vhdl/block4.vhd	blockB	gates	DC	Y
spec0/vhdl/block5.vhd	blockB	behavioral	BC	Y
spec0/vhdl/block6.vhd	blockB	rtl	DC	Y
spec0/vhdl/block7.mcl	blockC	mcl	MC	Y
spec0/vhdl/block8.mcl	blockC	mcl	MC	Y

#[End FILESPEC]

Figure 17. Example Design File

```
set_max_power 3mW
set_max_area 20000
clk_per_clk1 = 25
create_clock  -name clk1 –period 25ns –waveform {0 12.5}
set_input_delay  $clk_per_clk1*.1 –clock clk1 <input name>
set_output_delay $clk_per_clk1*.4 –clock clk1 <output name>
clk_per_clk2 = 50
create_clock  -name clk2 –period 50ns –waveform {0 25}
set_input_delay  $clk_per_clk2*.1 –clock clk2 <input name>
set_output_delay $clk_per_clk2*.4 –clock clk2 <output name>
```

Figure 18. Example Top Constraints File

The following subsections cover the various flows of different types of design source objects or design requirements that are anticipated in this tool environment. Reference Figure 16 for the appropriate entry point into the POMT flow.

It is worth noting that in each of the cases described in the following subsections, additional data, which may be needed to explore alternative architectures and complete a specific design implementation, is supported through customization or modification of the SDE script modules. DesignWare libraries provide basic building blocks of design objects, such as arithmetic functions, as well as data steering and decoder functions to allow for the most optimized logic to be created. The ASIC or FPGA library selection is performed through the SDE in conjunction with the synthesis tools.

3.2.1.2.2.1 Simulatable Specification Data Containing Top-Level Generic Design Requirements

The simulatable specification parser extracts top-level generic design requirements, converts appropriate requirements into Synopsys synthesis formats, and outputs these constraints into the TCF. The top-level design requirements may include information related to clock and input/output (I/O) timing, area, power and component technologies. This data output by the parser is then set up for SDE by the SimSpec2SDESetup tool. Synopsys has provided a complete specification for the simulatable specification parser Requirements and has developed the SimSpec2SDESetup Tool. The developed parser tool was developed for operation on PVS code (VSPEC) and is not currently available with SLDL (Rosetta), but is seen as a low risk translation.

3.2.1.2.2.2 Simulatable Specification Data Containing RTL Code Reference

In addition to the TCF, the simulatable specification parser creates a design file in FileSpec format suitable for submission into the SDE. The SDE, through the SimSpec2SDESetup tool, processes the information, as appropriate, along with constraint data, to drive the Design

Compiler (DC) or FPGA compiler synthesis tool as well as other Synopsys and third party tools integrated into SDE as desired.

3.2.1.2.2.3 Simulatable Specification Data Containing Behavioral Code Reference

The simulatable specification parser creates a design file representing the VHDL behavioral source suitable for submission into the SDE. The SDE, through the SimSpec2SDESetup tool, processes the information, as appropriate, along with constraint data to drive the Behavioral Compiler (BC) synthesis tool. The BC creates a VHDL RTL source representation of the design. The SDE is then used to process the VHDL RTL source information, as appropriate, along with constraint data to drive the DC or FPGA compiler synthesis tool as well as other Synopsys and third party tools integrated into SDE as desired.

3.2.1.2.2.4 Simulatable Specification Data Containing References GL Code

The simulatable specification parser creates a design file representing the GL source suitable for submission into the SDE. The SDE, through the SimSpec2SDESetup tool, processes the information, as appropriate, along with constraint data to drive the DC or FPGA compiler synthesis tool as well as other Synopsys and third party tools integrated into SDE as desired. This flow provides an easy, effective way for new silicon vendor libraries to be explored and/or targeted for the current physical realization of the design.

3.2.1.2.2.5 Simulatable Specification Data Containing References to Behavioral, RTL or GL

The simulatable specification parser creates a design file representing the modules of the design and their various levels of abstraction. The SDE, through the SimSpec2SDESetup tool, processes the files to drive the appropriate synthesis and design tools as required for each file type.

3.2.1.2.2.6 Behavioral or Graphical Design Environment Data

For design source data originating in a behavioral or graphical design representation, the tool (such as Mentor Graphics' Renoir) must provide an ASCII output representation of the design in VHDL format suitable for processing by Synopsys BC or DC. The SDE will accept this ASCII VHDL format information as described in the design file and drive the BC, DC or other tools as appropriate.

3.2.1.2.2.7 Stand-alone Design Source Data

The SDE accepts source data not embedded or pointed to by the simulatable specification. When stand-alone data is used, the SDE processes the data as directed, along with any constraint information provided, to drive the appropriate tools.

3.2.1.2.2.8 SDE Interface with Other Design Tools

The SDE provides user access to the script modules used to drive the Synopsys tools. With this access, the user can customize the modules for interfacing with non-Synopsys-supported design tools.

3.2.1.2.2.9 SDE Custom Interfacing with Synopsys Design Tools

The SDE provides user access to the script modules used to drive the Synopsys tools. With this access, the user can customize the modules for deriving advanced or customized results from the Synopsys design tools.

3.2.1.2.3 Additional Capabilities and Benefits of the BPR Approach

In addition to the capabilities referenced above, the use of SDE in the POMT program provides the following benefits:

- It is estimated that up to 48 percent of a design engineer's time can be spent in tool flow, environment, and integration issues. When using SDE the design team need not be expert toolsmiths on all the tools required in the design flow. The SDE user need only specify the flow order, associated compilation strategies and pertinent design information, and SDE creates the scripts to drive the tools in the flow. The user need only run these scripts as desired to support the targeted flow. Thus, the use of SDE significantly reduces the learning curve associated with the many point tools of the overall design flow and can cut the time spent on integration and flow issues by as much as 50 percent. The SDE also provides the flexibility to enable the expert toolsmith to customize the tool scripts and constraints as desired.

- In addition to providing general flow support for behavioral RTL, the SDE, via BC, assists in exploring alternative architectures for the realization of the desired function through the use of behavioral design exploration. Such exploration capabilities enable the designer to trade off throughput and area, enabling up to a 48 percent reduction in area or up to an 80 percent improvement in throughput. In addition, the BC can provide up to a 10x speed improvement in functional simulation and debug.

- The SDE provides graphical tools for viewing the results of various architectural implementations as well as generates design reports to evaluate the alternative architectures and component library specific solutions.

- Through the integration of VHDL simulators, SDE provides for automated RTL regressions, given the user-defined test bench. That is, once the test bench has been developed and supplied to SDE and the associated regression setup completed, SDE automates the running of RTL regressions and provides reports concerning their success or failure.

- Via its automated setup procedures, SDE enables the user to quickly switch between different silicon vendor libraries, thus assisting in the re-targeting and exploration effort while providing appropriate documentation as to the technology used and results obtained.

- The overall SDE environment effectively documents the design flow, source files, constraints, scripts, and individual steps run to produce the resulting design. Such

information can be efficiently stored and communicated via SDE's archive and test case generation capabilities.

- SDE already supports a comprehensive list of industry standard tools while providing the opportunity for others to be integrated as needed.

3.2.1.2.4 High Level Benefits of BPR Approach

The BPR effort and SimSpec/SDE embodiment leverages proven technology and industry standards for design creation and synthesis as well as proven technology in the area of behavioral synthesis. The SDE was extended and used to tie the database formats defined by the simulatable specification to existing standards for inputs to commercial tools. This approach minimized technical risk, lowered development cost, automated the flow-through of critical design information, and leveraged the commercially available SDE along with commercial CAD products. In addition, the open nature of SDE allows for the integration of additional tools as they are released to the market. Thus, the SimSpec/SDE flow can be adopted as time passes to keep pace with the best in class design, reuse, and obsolescence management practices as advancements in these areas are realized.

Furthermore, the integration of leading technology tools into SDE reduces the learning curve associated with the overall design flow while allowing expert users the freedom and flexibility to customize the environment as desired. The use of the BC provides a reduction in time-to-architecture of up to 50 to 80 percent. In addition, the use of SDE increases the productivity of the design development team and has been found to reduce the overall time-to-netlist by as much as 60 percent. The SimSpec/SDE environment also provides a comprehensive means of capturing the flow, methodology, and inputs used to achieve the particular implementation of the design at hand, thus ensuring repeatability and significantly reducing re-targeting effort.

Risk reduction was accomplished in this program through two key approaches. First, the use of existing, released, mature products to solve a high percentage of the technical challenges meant that fewer technical challenges needed to be overcome. Second, a modular approach to conversion of the design object representation with multiple opportunities to track the flow-down of all requirements and elements of the product implementation data set will allow the design development team to track and debug problems quickly.

This approach was arrived at based on experience in solving various tool interface and design object representation problems for a wide variety of customers and a wide variety of design requirements.

3.2.2 DVTG Methodology

The need to improve the development of the verification environment is becoming increasingly important. A survey by *ISD magazine* showed that more time is being spent on verification than is being spent on design. Design managers say that from 50 to 75 percent of the total time

is spent on verification. Add to that the fact that over the last few years, HDL-based design and synthesis has become the standard design flow. This has allowed designers to make huge increases in productivity, able to design very complex systems in only a few months. But as designs grow larger and larger, there has been no significant advance in the area of functional verification tools. The real challenge is to put a correct HDL description into synthesis. If we look at the total time spent on verification, over 50 percent is spent building up the test bench environment and developing the tests. So, verification teams are spending a lot of time before they even start simulation. Also, since design complexity is growing by Moore's Law, and the quantity of stimulus is also increasing exponentially, we could say that the verification problem is growing by Moore's Law squared. Figure 19 depicts the increasing role of verification in the design environment.

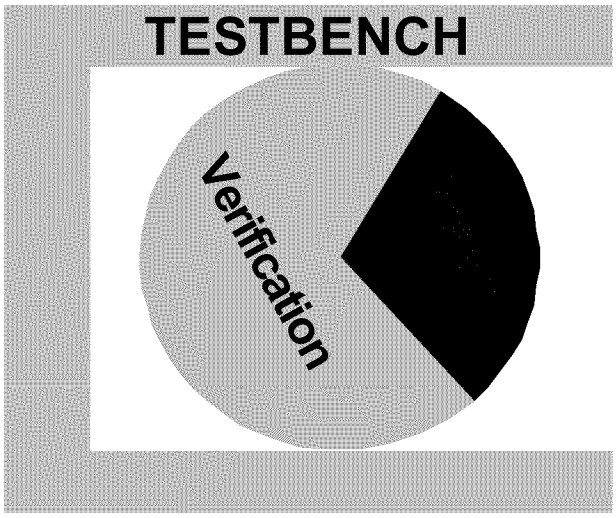

Figure 19. The Verification Problem

3.2.2.1 Goals and Objective

The University of Cincinnati has developed a set of tools which operate within the reengineering tool environment as a plug-in and provide partial automation support to the product reengineering methodology in the area of the development of design verification tests for digital electronic products. These tests are in the form of test benches and accompanying test vectors. These tests were developed based upon formal product requirements models written in the Rosetta language. More specifically, the requirements model is represented by a requirements facet written in the SLDL, which is also known as Rosetta. These tools contribute both to the significant reduction in the effort required to develop a new product design as a part of the reengineering methodology and to the reduction in the opportunity for manually introduced errors and incomplete design verification. For a brief overview of Rosetta, please see section 9.

3.2.2.2 Implementation

The DVTG tool accepts as an input the formal requirements model for the product written in the SLDL/Rosetta language and contained in the simulatable specification for the product. The

DVTG tool partially automates the process of developing formal design verification tests to be used to test product implementations against the product requirements. The DVTG tool develops these tests in the form of abstract test vectors, which can then be translated into waveform and vector exchange (WAVES) test vectors for use with accompanying WAVES/VHDL test benches. The benefit of the DVTG tool is anticipated to significantly reduce the test engineering effort and reduce the risks associated with each reengineering pass. A high level view of the process is depicted in Figure 20. The steps to the process are as follows:

1. Specify the design's requirements
   - Rosetta is used to annotate a design (develop requirements facet)
   - Rosetta requirements are analyzed to ensure correctness (prove the requirements facet). This capability currently exists with VSPEC and it is possible to translate it to Rosetta.
2. Requirements are refined into a specification and implemented in hardware and/or software.
3. Test vectors are semi-automatically generated to determine correctness of refinement:
   - Derive the various inputs and their expected outputs from the Rosetta specification model.
4. Validate the design
   - Exercise the implementation with the specified input values using a simulation environment or test facility
   - Check if the implementation generates correct outputs.

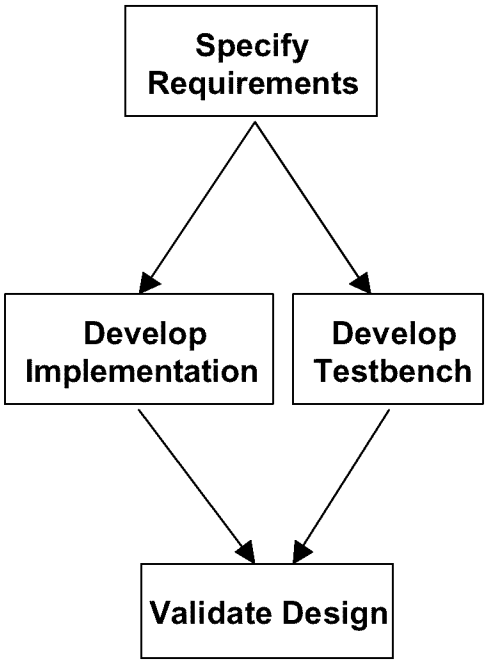

Figure 20. Validation Flow

In the flow described above, steps 2 and 3 can be accomplished concurrently. In step 4, the objective of validation is to determine the following:

- Did you build it right?
- Did you move from requirements to specification to implementation correctly?

Once you have validated the design with both the test bench and the implementation being derived from the requirements, there is a high degree of confidence that the design will work as intended.

Although Figure 20 provides a necessary view of the overall flow, the focus of DVTG resides within the development of the test bench step. Figure 21 provides a more detailed overview of how the DVTG tool fits within the environment.

Figure 21 depicts the flow for test vector generation. The primary inputs to the process include the requirements facet (which was formerly represented by VSPEC) and the test facet. Each of these are explored in greater detail later. Effectively, these facets represent the requirements of the design and the requirements of the test on that design. Together, these facets can be used to create test vectors that can eventually be used directly with a test bench, or converted into test vectors to be used within a specific simulation environment.

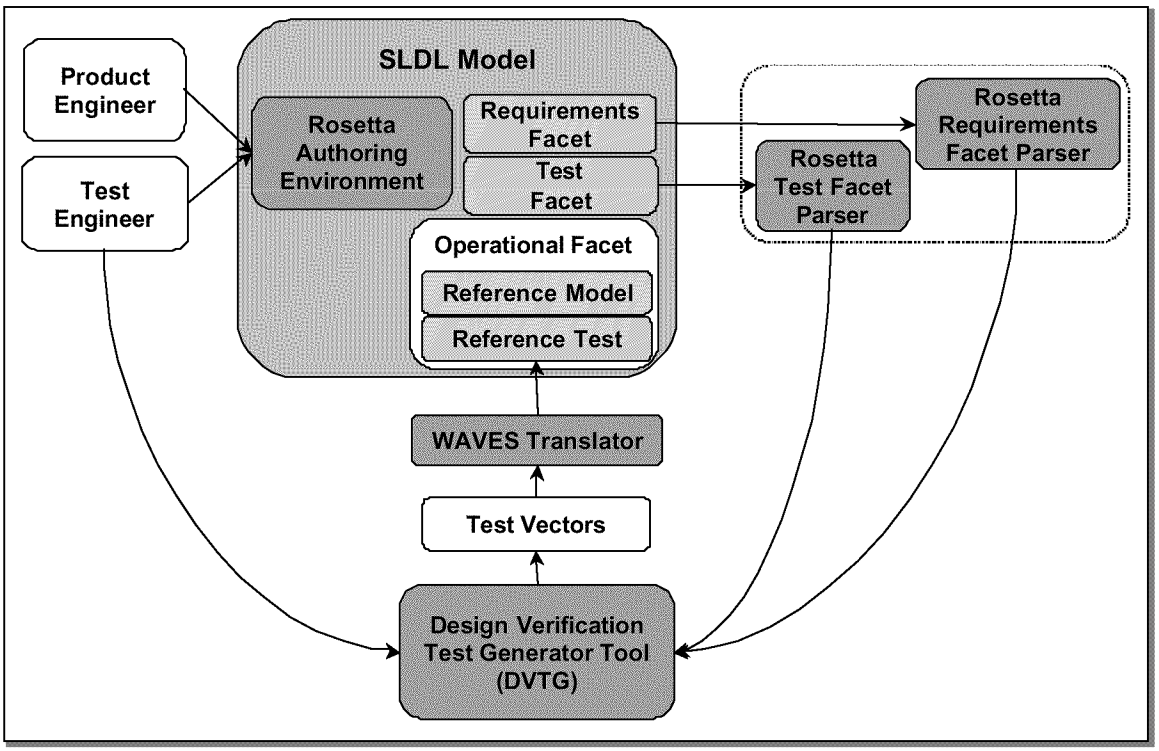

Figure 21. DVTG Tool Flow

A more detailed flow for the development of the test vectors within the DVTG tool is shown in Figure 22. For a more complete understanding of the DVTG methodology, an overview of the underlying logic within each of the blocks depicted in Figure 22 follows.

3.2.2.2.1 System Requirements

The primary inputs to the flow are the system and test requirements. The system requirements are represented in Rosetta by the requirements facet and are what would be formerly represented by the VSPEC specification.

3.2.2.2.2 Test Requirements

As mentioned earlier, the test requirements are a key input to the DVTG tool. The test requirements are represented in Rosetta by the test facet and indicate specific requirements for test implementation. Initially, they were limited to ranges and increment sizes on input variables. For example, the '*var* : *start* to *end* : steps of *count*' test requirement translates to the test input *var* starting with value *start*, ending with value *end*, in increments of *count*.

Recent enhancements have been made in this area. Test requirements are no longer limited to simple range and step specifications, but allow the user to specify the following:

- characteristics of input system signals
- coverage of requirements
- initialization codes used to set up test scenarios.

The DVTG capability allows the user to specify input signal characteristics in the traditional manner by specifying signal-noise characteristics, sample size and sample rate. Using calls to MATLAB™, DVTG generates signal and noise waveforms, samples each and adds samples to generate input to the device under test. The user may specify any waveform or noise source by either updating test requirements or augmenting the functions available to the DVTG system.

Coverage limitations are used to specify how an input variable's range is to be covered in the testing process. Currently, coverage requirements are specified as an initial and final value with a nonzero step size. The DVTG also automatically generates tests to examine boundary conditions in the required range.

Frequently it is necessary to drive a system to a particular state prior to executing a test vector. The DVTG supports this by allowing the user to write test vectors that generate the desired system state. These vectors are associated with the resulting state value and are automatically included in generated vector sets when state initialization is required. The DVTG also provides mechanisms for looping through test vectors and avoiding initialization prior to vector execution.

3.2.2.2.3 Test Scenario

The test scenario is an abstract specification of how the design unit will be tested. It indicates classes of tests to be performed, including what the requirements coverage is and what the preconditions and post conditions are for correct operation. The test scenario is generated using specification-based testing techniques and includes both black box and white box testing. In black box testing, it tests for precondition and post condition coverage, whereas in white box testing, it pulls the preconditions and post conditions apart to more fully exercise the model for more complete specification coverage. An example of this would be testing both the "then" and "else" conditions in an "if-then-else" statement. One key factor associated with test scenarios is that they cover only requirements level issues; implementation issues are deferred. This

approach is more abstract than traditional automated test vector generation (ATVG) systems and does not perform "stuck at" or glitch detection.

Test scenarios are generated from logical operators in the Rosetta specification. These logical operators indicate the types of the scenarios and how many scenarios must be tested. Table 4 portrays some representative examples.

Table 4. Logical Operators and Generated Test Scenarios

Specification Form	Scenarios to Test
P(x) and Q(y)	(P(x) = true) and (Q(y) = true)
P(x) or Q(y)	P(x) = true Q(y) = true
Not P(x), then Q, else R endif	P(x) = true and Q P(x) = false and R

In the "P(x) or Q(y)" specification form, when evaluating the "P(x)=true" scenario, Q(y) is considered a "don't care" condition. Likewise, when evaluating "Q(y) = true", P(x) is a "don't care" condition. The logical operators (and, or, not, if then else) may also be nested and the scenarios to test will follow the nesting.

When a logical predicate is asserted, a class of tests is generated. For example, the following classes of tests are generated:

- sensitive to P(x) generates a test scenario to assure P(x) activates the component
- requires P(x) generates a test scenario to assure P(x) is true upon initiation
- ensures P(x) generates a test scenario to assure P(x) is true upon termination.

3.2.2.2.4 Abstract Test Vectors

The primary outputs of the DVTG tool are the abstract test vectors, which specify test driving values and expected output conditions. These test vectors are the actual input values generated for a simulator and are independent from any specific description language. However, the test vectors can be translated into whatever simulation environment necessary to test the implementation (e.g., VHDL/WAVES format if the implementation and test bench are represented in VHDL).

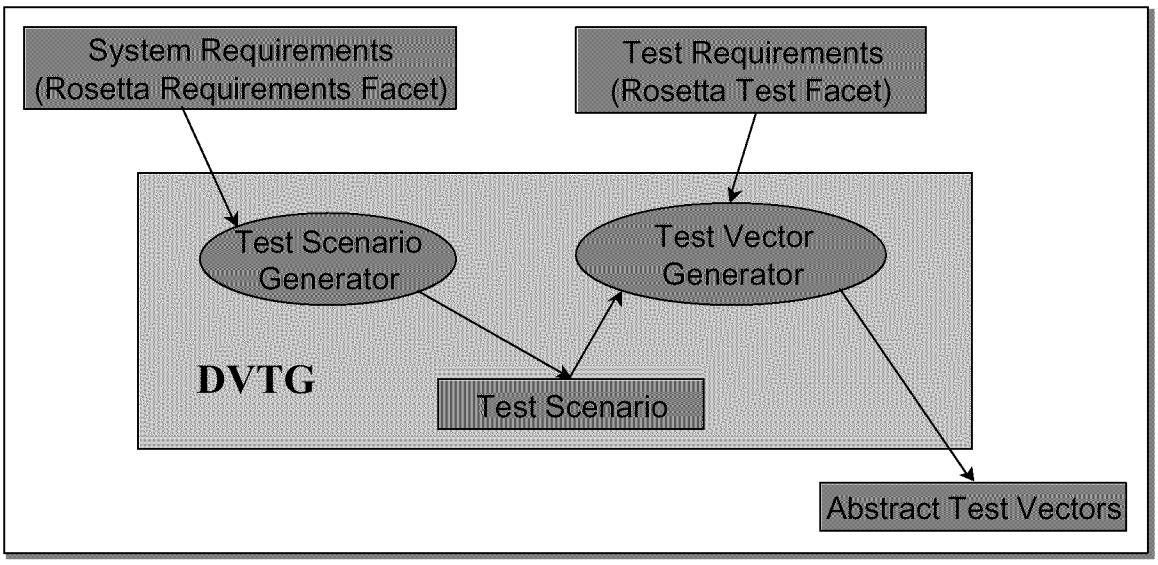

Figure 22. DVTG Information Flow

3.2.2.3 DVTG Example

At this point, a simple example may be beneficial to demonstrate the process. The code in Figure 23 is representative SLDL (Rosetta) code for a simple schmidt-trigger.

```
entity schmidt-trigger
  port(j:in real; o:out bit);
state b:bit;
requires j>0.0 and j<5.0;
ensures
  if j<1.0 then state'post=0
    else if j>4.0 then state'post=1
    else state'post=state
    endif
  AND
  o=state;
end schmidt-trigger;
```

Figure 23. Rosetta Code Example

The test scenarios generated from the above example include the following:

- j<1.0 and state = 0 and o=state
- j>=1.0 and j>4.0 and state = 1 and o=state
- j>=1.0 and j<=4.0 and state'post = state and o=state.

In these scenarios, conditions on inputs indicate driven values, and conditions on outputs indicate checks or tests that need to be made.

Evaluating the first test scenario (j<1.0 and state = 0 and o=state) and applying test requirement, j : 0.0 to 5.0 : steps of 0.2, yields the following test vectors.

- j = 0.0 and state=0 and o=state
- j = 0.2 and state=0 and o=state
- j = 0.4 and state=0 and o=state
- j = 0.6 and state=0 and o=state
- j = 0.8 and state=0 and o=state.

3.2.2.4 Benefits of DVTG

As discussed previously, the creation of the test bench and test vectors to drive the simulation has become an increasingly difficult task. The DVTG addresses a portion of this problem by providing the capability to partially automate the test vector generation process. Additional benefits of this approach include the following:

- Tests for incomplete, inconsistent or misinterpreted requirements. This is done prior to the DVTG tool within the proving capabilities of the requirements facet. Proving is accomplished through the use of Prototype Verification System (PVS), and was previously demonstrated with VSPEC. This is an anticipated future capability of Rosetta.
- Ensures better conformance of implementation to original requirements through the generation of test vectors directly from the product's specifications as opposed to an implementation of the product.
- The outputs are independent of specific simulation or specification environments and can easily be converted into traditional simulation and testing environments.
- Significantly reduces development and redevelopment time for test cases both initially and even more so when requirements change.
- Significantly reduces the impact of manual error injection into the test development process.

3.2.3 Integrated Tool Flow

In the discussion of the tool methodology so far, the focus has been on the requirements and capabilities of individual tools. The methodology also needs to address how the individual tools will work together and how the tools fit into the overall design flow. This will be handled in an incremental fashion by discussing how the tools fit into different pieces of the overall information flow.

One of the main components for either of the flows that we will discuss will be the design environment that will be used to drive the flow. In each of the flows discussed, it is assumed that a graphical design environment (such as Mentor Graphics' Renoir) will be used.

3.2.3.1 Requirements Specification Proving Flow

The first flow discussed here is associated with proving the requirements specification as shown in Figure 24. After the creation of the requirements facet, PVS code is generated. The PVS generation involves the creation of the PVS model and the PVS proof generation code that are

input to the PVS prover, which in turn provides the results of the proving exercise. This path was demonstrated with VSPEC as the requirements specification language. Although the figure shows a Rosetta flow, it is not currently available with SLDL, but is seen as a low risk translation from VSPEC.

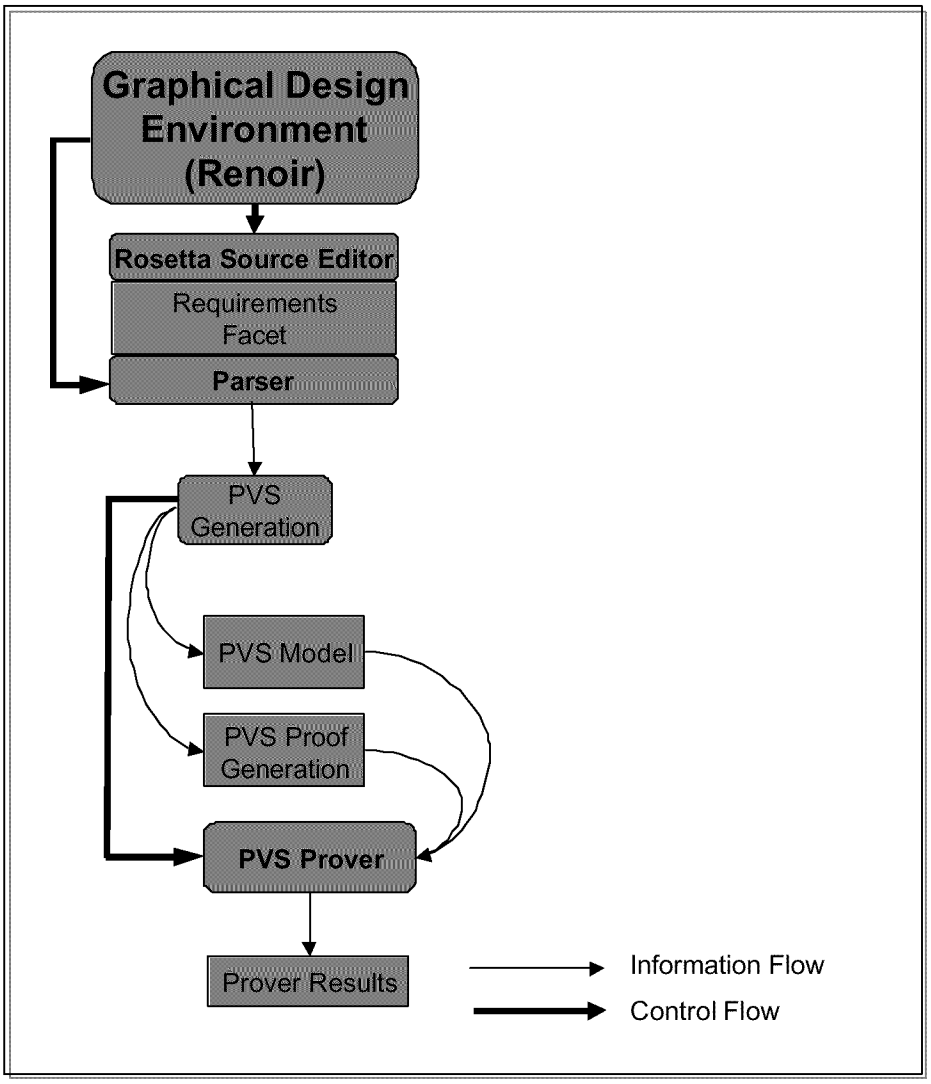

Figure 24. Requirements Specification Proving Flow

There are several classes of errors that proving catches. One class is when requirements either directly or indirectly conflict. It is also capable of catching problems with non-functional requirements like power constraints that are not considered in simulation processes. Proving also deals well with incomplete specifications that cannot be simulated or systems where simulation cannot generate enough throughput to catch an error. For example, if an error appears every 10^{12} states, simulation is not going to catch it, but theorem proving or model checking will.

3.2.3.2 Representative Virtual Prototype Flow

The DVTG is introduced into the flow as part of the overall development of a virtual prototype as depicted in Figure 25. In the interest of not making the diagram too cluttered, the proving path has been removed. In this flow, in addition to the creation of a requirements facet, a test facet is also created for the purpose of developing test vectors. The two facets feed the DVTG tool, which proceeds to develop the abstract test vectors described previously. In this representative example, these abstract test vectors are then converted into VHDL/WAVES compatible test vectors for use in a VHDL simulation environment.

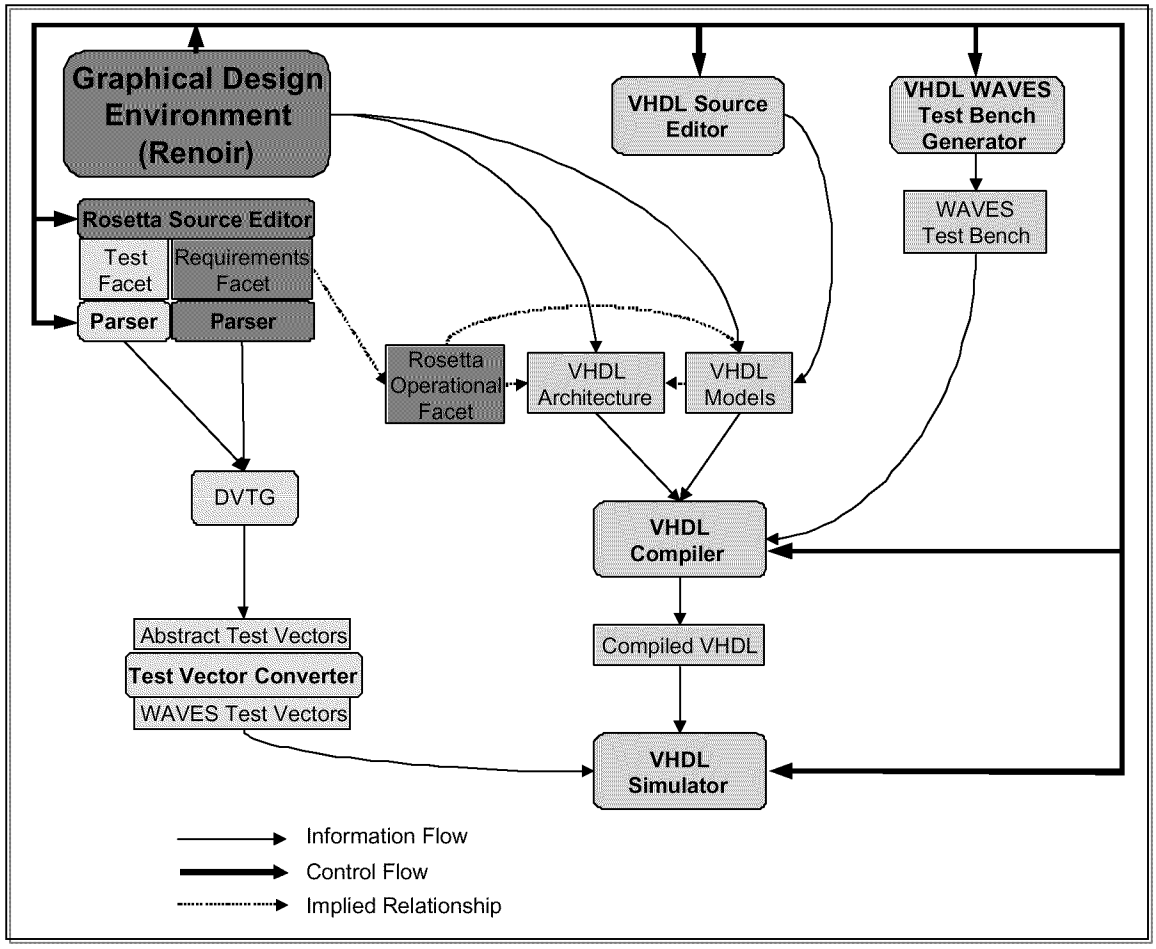

Figure 25. Representative Virtual Prototype Flow

Previous paragraphs detailed the flow through the generation of test vectors for use within the simulation environment. There must also be a design or implementation that is exercised by the vectors. This includes both an implementation of the design and the reference test.

The implementation of the design is also referred to as the operational facet and is the culmination of developing a design that is both simulatable and synthesizable and is based on the requirements facet. Most likely, there will be some functional decomposition of the

requirements facet to partition the design into more manageable design units. It is important that the requirements are also partitioned, flowing down to the individual design units. Section 3.2.3.2.1 briefly discusses this design decomposition.

3.2.3.2.1 Design Decomposition

The requirements model, represented by the Rosetta requirements facet, may or may not be decomposed hierarchically. In many cases, the model is decomposed to allow for flowing requirements down to the component level or to break the requirements up into more manageable pieces. If the design is decomposed into multiple models, those models will have an architecture associated with them determining how the various models are connected. It is important to maintain relationships between the requirements model and the design's implementation (operational facet) when decomposing. Figure 26 illustrates an example functional decomposition maintaining the implied relationship between the SLDL (Rosetta models) and the VHDL.

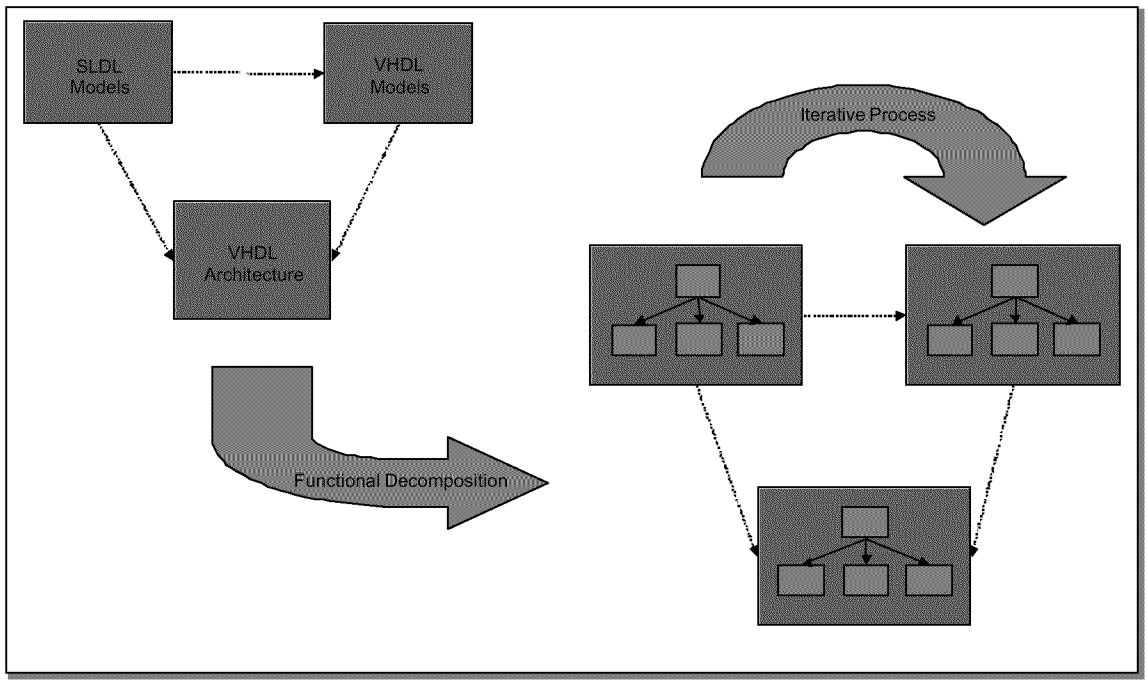

Figure 26. Maintaining Relationships Hierarchically

Figure 27 provides a similar view of how the requirements are decomposed along with the operational facet (reference design) and reference test, decomposing the design from the system level down to the line replaceable unit (LRU) level.

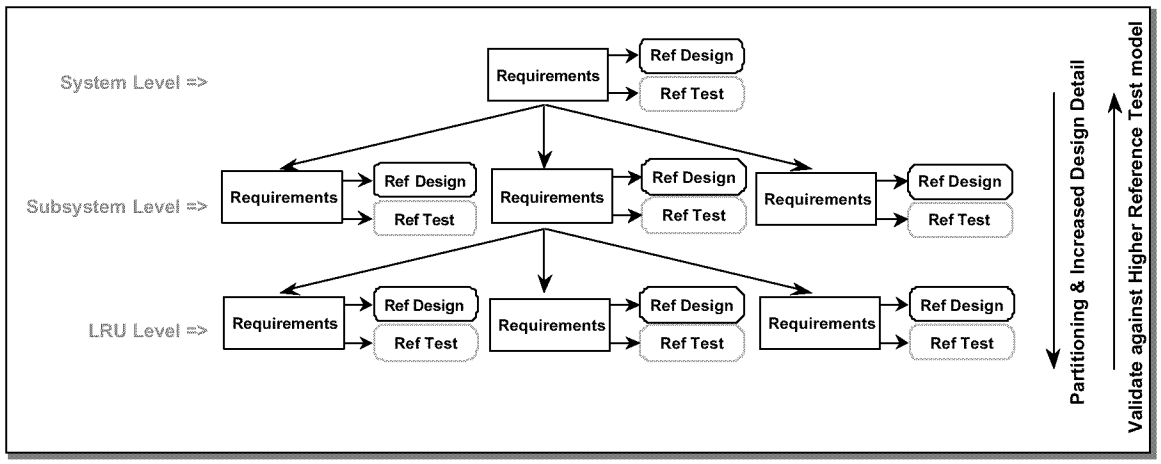

Figure 27. Design Decomposition

3.2.3.3 Representative Synthesis Flow

The requirements proving flow and the virtual prototype flow have already been discussed. What remains to be discussed is a flow that takes the design through synthesis and into hardware. Figure 28 provides an example flow for this process. The highlights of this flow are the SDE setup tool and the VHDL synthesis step. The SDE setup tool parses out the requirements pertaining to synthesis from the requirements facet and incorporates them into the SDE environment. For more detail on this process, please refer to section 3.2.1 along with Figure 15 and Figure 16.

3.2.3.3.1 Requirements Facet Parser

The need to generate Synopsys synthesis constraints was addressed by enhancing the parser for the requirements facet. The original VSPEC language supported specification of both functional requirements and performance constraints. To support Synopsys constraint generation, the language was enhanced to represent new constraint types, and the parser was modified to generate input files for the Synopsys synthesis toolset.

Supporting Synopsys synthesis required providing information on power consumption and area limitations as well as clocking issues. Power and area limitations were already supported by the language and parser. Thus, generating Synopsys constraint inputs was a matter of reformatting the existing constraints. Clock descriptions and constraints required adding clock definitions to the language. Because VSPEC explicitly supports extensions of this kind, the update was accomplished by updating a configuration file.

The following VSPEC constraint section defines the format for power and area constraints as well as defines two clocks showing the format for constraint representation:

```
power <= 3mW
size <= 100um * 200um
clock is name=clk1, period=25ns, waveform=(0 12.5)
```

```
// The following are assumed to be in clk1 domain and the delay is
// a percentage of the clock period above.
input delay <input name> <percent of period>
output delay <output name> <percent of period>

clock is name=clk2, period=50ns, waveform=(0 25)
// The following are assumed to be in clk2 domain and the delay is
// a percentage of the clock period above.
input delay <input name> <percent of period>
output delay <output name> <percent of period>
```

The example above is translated into the following Synopsys constraint format:

```
set_max_power 3mW
set_max_area 20000
clk_per_clk1 = 25
create_clock  -name clk1 –period 25ns –waveform {0 12.5}
set_input_delay  $clk_per_clk1*.1 –clock clk <input name>
set_output_delay $clk_per_clk1*.4 –clock clk <output name>
clk_per_clk2 = 50
create_clock  -name clk2 –period 50ns –waveform {0 25}
set_input_delay  $clk_per_clk2*.1 –clock clk <input name>
set_output_delay $clk_per_clk2*.4 –clock clk <output name>
```

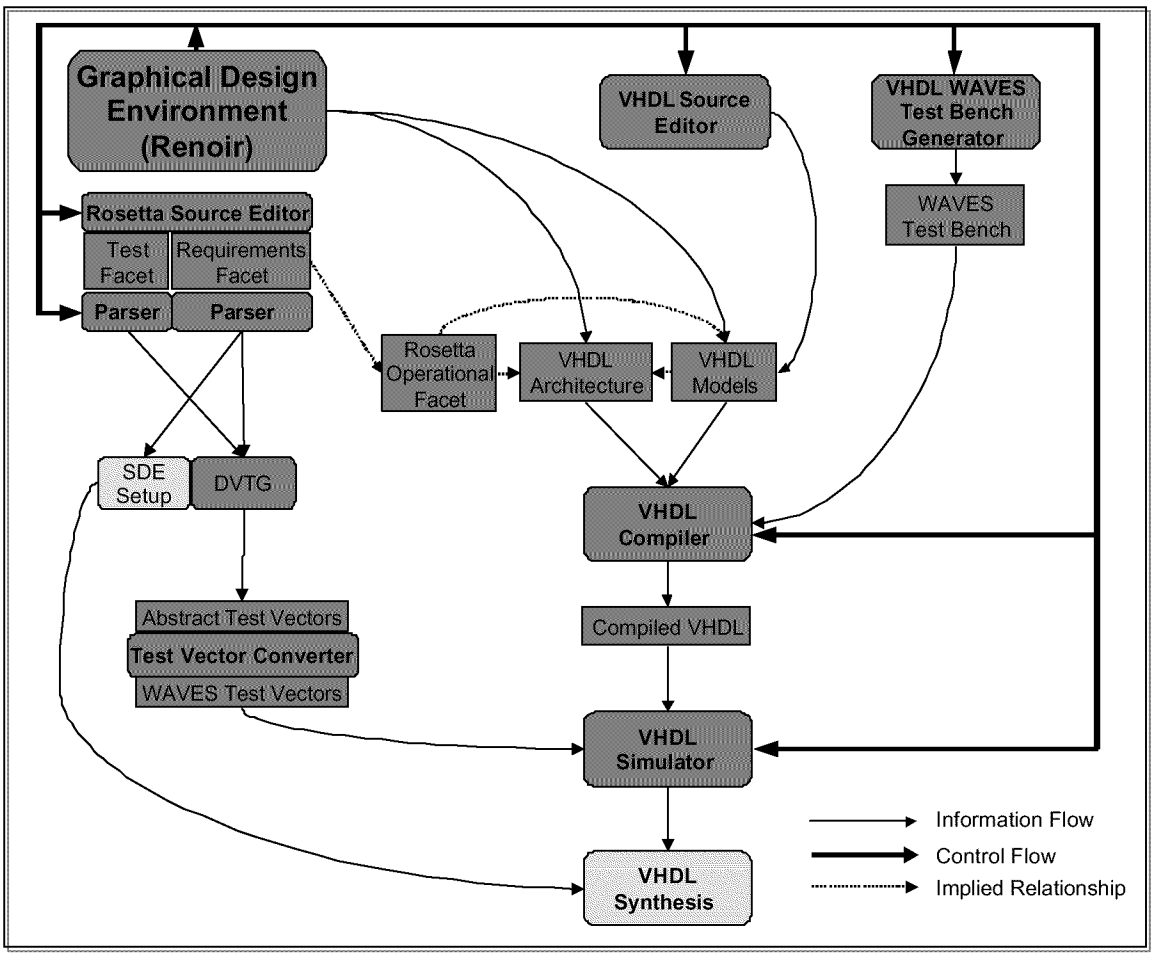

Figure 28. Representative Synthesis Flow

Section 3 has provided a description of the methodology, tools and how they fit together. The next steps were to perform demonstrations implementing the methodology and tools, and collect metrics to determine how effective the tools and methodology were. Section 4 provides a description of the various demonstrations that were performed, and section 5 provides the results of the demonstrations.

4. POMT DEMONSTRATION

This section describes the demonstration portion of the POMT program and provides an overview of the different demonstration targets for both the BPR tool developed by Synopsys, as well as the DVTG tool developed by the University of Cincinnati.

4.1 Purpose

The demonstrations serve two purposes. They test the methodology and the tools as well as illustrate the capabilities of the methodology and tools. Initially, the demonstrations were used during the development of the BPR and DVTG tools for debug and test purposes. Upon completion of the BPR and DVTG tools, the demonstrations were used to illustrate the capabilities of the individual tools as well as how the individual tools tie in to the methodology.

4.2 Objective

No single demonstration, within this efforts scope and budget, was capable of demonstrating a significant portion of the methodology and tools. Therefore, multiple, limited-scope examples were developed to test different aspects of the methodology and tools. It is also a goal to demonstrate that the methodology and tools support both commercial and military design environments and design scopes ranging from relatively small to robust. These goals included focusing at least one demonstration on each of the following:

- a "releasable" design, allowing for the distribution of the demonstration
- a military application
- a design scope which is of significant size but does not require major development due to limited time and budget constraints
- a mix of behavioral VHDL and RTL VHDL.

4.3 Demonstration Targets

There were multiple demonstration targets, and each demonstration focused on a different tool capability. These included several demonstrations focusing solely on the BPR tool and others that exercised both the BPR and DVTG tools.

The demonstrations focused on the BPR tool include the following:

- Alarm clock
- Core
- Behavioral Core.

The demonstrations that exercise both the BPR and DVTG tools include the following:

- Behavioral alarm clock
- Satellite communications (SATCOM) application

A brief overview for each demonstration is provided in the following sections.

4.3.1 Alarm Clock Design

The alarm clock demonstration is a very simple design that implements an alarm clock with hierarchical RTL VHDL. The purpose of this design is to demonstrate the POMT flow through SDE that utilizes DC.

4.3.2 Behavioral Alarm Clock Design

This design is very similar to the alarm clock design but has a behavioral module added in order to demonstrate the power of the BC and architecture exploration within SDE. In addition, this is a simple design that demonstrates the POMT flow from SimSpec2SDESetup through SDE utilizing the FPGA compiler. The VHDL code and both the requirements and test facets are provided in section 10.

4.3.3 Core Design

This design is intended as a supplement to the alarm clock designs. The purpose of this design is to provide a larger, more complex design that exercises a broader set of features within SDE. This design represents modules of varied levels of abstraction and include behavioral HDL.

4.3.4 Behavioral Core Design

This design is very similar to the Core design with the addition of a more complex behavioral module, an automatic gain control (AGC) loop. This design includes a scheduling script to demonstrate the architectural trade-off capabilities of the BC and a self-checking test bench with graphical output review.

Note that the Core and behavioral Core designs are intended to be supplemental in nature relative to the alarm clock designs in order to cover areas in which the alarm clock designs do not provide sufficient demonstration capabilities. These designs do not have an associated SLDL model developed for them and focus primarily on testing capabilities of the Synopsys tools that are not being utilized by the other demonstration examples.

4.3.5 SATCOM Application

The SATCOM application demonstration implements a basic processing thread of some of the preprocessing functions required for a user to enter and communicate in a SATCOM time division multiple access (TDMA) network. Additionally, the demonstration application implements preprocessing functions required for receiving single-access channels.

In TDMA systems, messages from several users are interlaced in time. The data from each user are transmitted in time intervals called slots, with a number of time slots comprising a frame. Within a single time slot only one user may transmit or receive data, so each user occupies a cyclically repeating time slot. Each time slot consists of a preamble and information bits modulated onto a RF carrier. The preamble is used to provide identification and allow synchronization of the time slot at the intended receiver. Guard times are placed between each user's transmission burst to minimize cross talk between channels.

In order to communicate in a TDMA network, the user must be able to detect a transmitted unique word preamble in the presence of additive white Gaussian noise. This is the requirement for the demonstration preprocessor. The functions necessary to detect this preamble include carrier recovery, bit synchronization, and correlation with a unique word. The unique word type, modulation type, modulation rate, and other parameters are variable control inputs to the preprocessor. The preprocessor decodes the control inputs to determine the type of message to expect. Figure 29 shows a general block diagram of the preprocessor operation.

A critical component of the demonstration is the generation of input vectors to test the performance of the preprocessor against the system requirements. The input test vectors include the control inputs and sets of digitized input message waveforms with varying modulation types, modulation rates, Doppler frequency shifts, signal-to-noise ratios, etc. The demonstration should also provide a meaningful measurement of preprocessor performance versus the requirements.

The algorithms developed for this demonstration are potentially applicable to future use on JSF and/or Comanche CNI systems. This design provides the most applicable vehicle for a military demonstration and is the most robust of the examples. It is an end-to-end demonstration, meaning that an SLDL model was developed for the design, and it was taken through to hardware synthesis. Both the requirements and test facets are available at the Rosetta web page (www.sldl.org). Unfortunately, due to the proprietary nature of the implementation (operational facet), the code for this demonstration will not be made available.

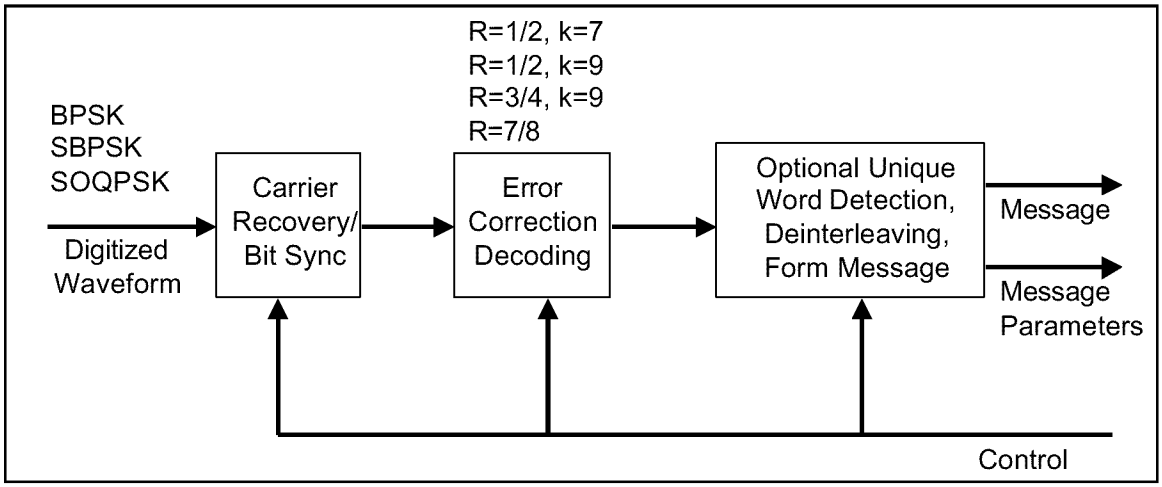

Figure 29. SATCOM Preprocessor Block Diagram

4.4 Demonstration Process

The detailed process utilized in conducting the demonstrations for the various targets follows the methodology described in section 3. The process is briefly described in the following subparagraphs for overview purposes and is a specific implementation of the POMT methodology, given the tools and techniques available.

The first step of the process for the demonstrations that tested both the BPR and DVTG tools was to develop a simulatable specification. The simulatable specification began with the customer's requirements, which were then translated into the requirements facet to be used as the basis for the design of the product. In addition to the requirements facet, a test facet was also developed. The test facet was used in conjunction with the requirements facet within the DVTG tool for development of test vectors. These test vectors were then used to validate the operational facet, which was the actual product implementation.

The demonstration development consisted of the steps identified in Figure 5. A slightly more detailed depiction of the process is presented in Figure 6. For an explanation of the process described in Figure 5 and Figure 6, see section 3.1.2.

4.4.1 Demonstration Design Flow

As described previously, Figure 5 and Figure 6 describe the methodology for design development. In this section, the flow for the demonstration approach is defined. Figure 30 represents a more sequential view than the one presented as part of the methodology design flow. It is more representative of the flow that the POMT demonstrations, utilizing both the BPR and DVTG tools, have taken. The demonstrations focus solely on the BPR, concentrated on the Synthesize Operational Facet step. Each of the facets are discussed in the following sections.

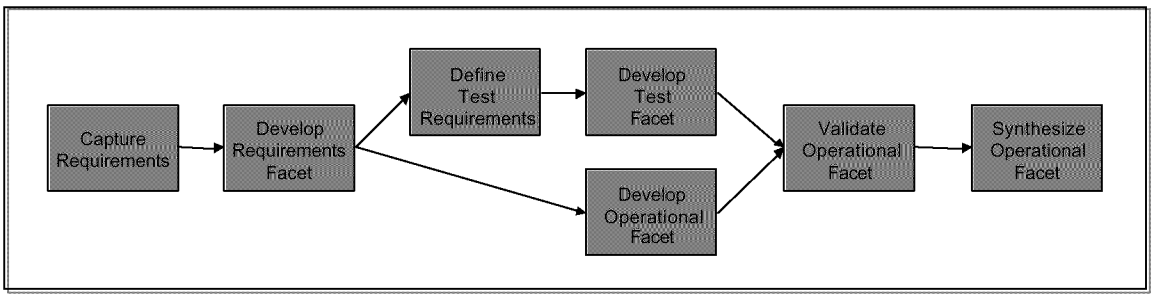

Figure 30. Demonstration Design Flow

The first step of the process was to capture the requirements. This was initially represented with a paper specification. Once the requirements had been captured, the next step was to develop the requirements facet, which was represented in a Rosetta requirements facet. At this point, two parallel activities took place. With the requirements facet as the baseline, the operational facet and test bench were developed concurrently. The test bench included the task of defining the test requirements which was used to develop the test facet. With the test bench and operational facet in place, the operational facet was then validated by exercising it against the test bench and its associated test vectors. Discrepancies were addressed before the operational facet was synthesized in the final step. After the design was synthesized, the synthesized design was revalidated against the same test bench. This process is an elaboration of the process described in Figure 20. For an even more detailed description of the flow and how individual tools fit into that flow, see Figure 28.

4.4.1.1 Core Demonstration

As detailed in section 4.3, the Core demonstrations were not formal demonstrations and were focused entirely on the BPR tool. Their intent was to exercise the BPR tool with a more complex design than the alarm clock demonstration. This was accomplished by exercising more of the features utilized within the SDE, which is the basis of the BPR tool. The SATCOM application demonstration was considered robust enough, but was not far enough along in development to exercise these features. This was due to the shorter development cycle of the BPR tool. The Core demonstration was used instead to validate the BPR tool prior to the end of the Synopsys effort.

From the standpoint of Figure 30, both the Core and behavioral Core demonstrations exercised only the following three portions of the design flow:

- Develop operational facet
- Validate operational facet
- Synthesize operational facet

The validate operational facet portion of this flow was not very meaningful, however, due to the absence of a test bench.

4.4.1.2 Alarm Clock Demonstration

As detailed in section 4.3, the alarm clock demonstrations were developed to provide a demonstration of the flow. Only the behavioral alarm clock demonstration was utilized for an end-to-end demonstration of both the BPR and DVTG tools.

A requirements facet was developed for the alarm clock demonstration and applies to both the alarm clock and behavioral alarm clock designs. The VHDL code (operational facet) for the behavioral alarm clock demonstration as well as both the test and requirements facets are available in section 10.

Although not as robust as the SATCOM application demonstration, the behavioral alarm clock demonstration provides a simpler, and thereby easier-to-understand, model for both the flow itself as well as the representations of the various facets involved.

4.4.1.3 SATCOM Application Demonstration

The design flow used for the SATCOM application demonstration differed slightly from that shown in Figure 30. For validation purposes, the development of the test bench and operational facets included an additional step. A third party, high level, system simulation tool (SystemView) was used to develop algorithms and test vectors. The modified design flow is shown in Figure 31. Steps 1 through 4 of the following flow detail the key differences from the proposed POMT methodology:

1. The conceptual design, as simulated in SystemView, was validated against the test vectors generated within SystemView.
2. Once the conceptual design was validated, the VHDL implementation began, and the VHDL design was subsequently tested using the SystemView test vectors.
3. Once the DVTG test vectors were generated from the test requirements facet, they were used in place of the SystemView test vectors.
4. The DVTG test vectors were then compared with the SystemView test vectors to verify their correctness. More importantly, the results of the VHDL simulations were compared to validate the DVTG test vectors.
5. Once the DVTG test vectors were validated, they were used to test the operational facet, as described in section 4.4.1.

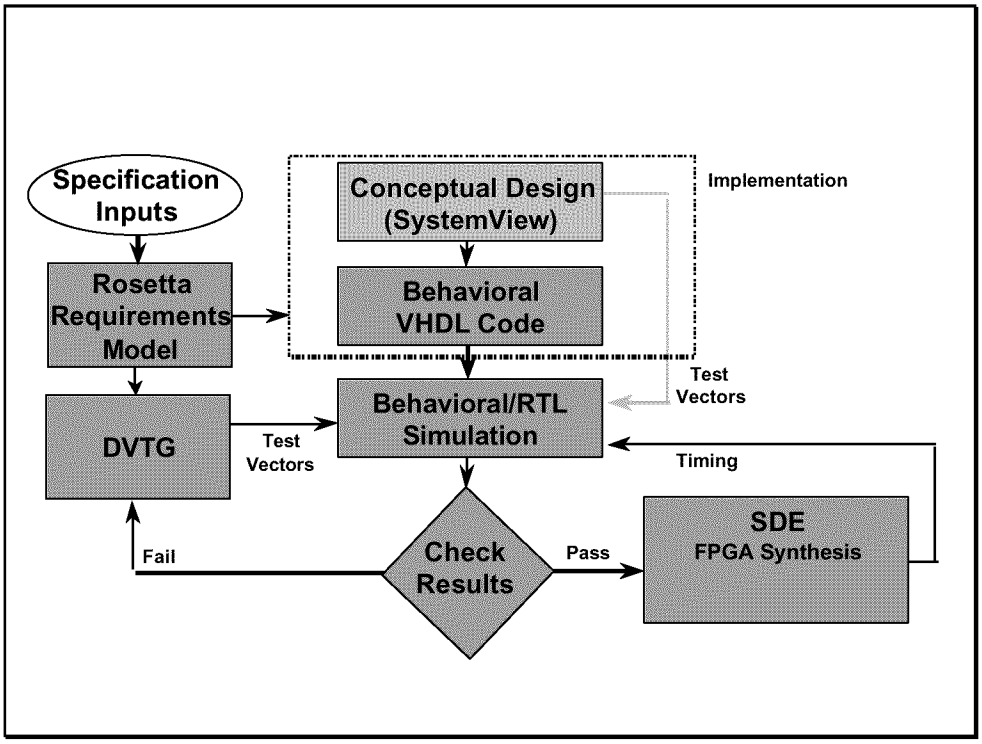

Figure 31. SATCOM Application Design Flow

4.4.2 Facet Development

As discussed previously, facets will need to be developed for the demonstrations. Table 5 depicts the facets that were developed for each of the demonstrations. The two Core demonstrations have only an operational facet associated with them, while the remaining three demonstrations also include the requirements and test facets. The same requirements facet and test facet were used for both of the alarm clock demonstrations. The only difference between these two demonstrations is in the implementation.

Table 5. Facet Development

	Requirements Facet	Test Facet	Operational Facet
Core			X
Behavioral Core			X
Alarm clock	X	X	X
Behavioral alarm clock			X
SATCOM application	X	X	X

4.5 Benefits Analysis

To support the benefits analysis activity, the baseline was the SATCOM application demonstration task. The main steps involved included collecting metrics and analyzing the collected metrics.

4.5.1 Collect Metrics

For the SATCOM application demonstration, supplemental time logs were maintained, documenting the time performed in accomplishing the various tasks. This information was collected during the performance of the case study in order to support the analysis activities.

4.5.2 Analyze Results

Following the completion of the case study, the collected metrics were analyzed and the demonstration was characterized in terms of its complexity so that the metrics could be extrapolated to larger applications. In addition, the people performing the case study were interviewed in order to identify shortcomings of the methodology and/or tools as well as suggestions for changes/enhancements in the methodology and/or tools. The analysis results identify what worked and how well, as well as what did not work and why.

5. RESULTS

In this section, both the BPR and DVTG tools as well as the demonstrations are evaluated relative to their initial requirements. It will detail what the requirements were and whether or not each requirement was met. If the requirement was met, it will describe how it was accomplished.

5.1 BPR Results

As can be seen from the implementation description discussed in section 3.2.1.2, the BPR tool effort meets all of the following defined objectives:

- An implementation-independent data set for the design is established and used to generate a physically realizable design.
- All levels of source code abstraction from behavioral to gate level netlists are supported.
- An environment is provided in which the best-practice design flows are established and can be modified as technology progresses.
- Widely used and accepted de-facto industry standard tools that use industry standard data exchange formats are used.
- Automation of re-targeting of designs to new silicon vendor libraries is supported.
- Effective documentation of the design flow, constraints, tools, and source files used is established to assist in reuse and higher level integration.

5.1.1 BPR Requirements and Their Fulfillment

This section discusses key requirements in the development of the BPR tool. Following is a list of the requirements and how these key requirements were met:

Requirement: Schematic level view.
Approach: The Synopsys tools provide an ability to present the final schematic view of any given design implementation. Additionally, the Synopsys tool set for behavioral design provides a mechanism for viewing the results of behavioral coding and architectures implied.

Requirement: Mapping entities and signals between behavioral representation and components of implementation.

Approach: The Synopsys tools carry forward, from (synthesizable) behavioral code to RTL and finally to a gate level netlist, the correct resulting pins. This capability will be maintained throughout the flow proposed.

Requirement: Generation of synthesizable VHDL packages for each custom component.
Approach: The SDE and Synopsys BC tool supports the generation of synthesizable VHDL packages for code accepted at the behavioral level. This capability is subject to the limitations of coding style supported by the Synopsys BC product.

Requirement: Generation of VHDL architecture for the module.
Approach: When provided the behavioral level representation (in VHDL) from the product implementation data set or simulatable specification, the SDE tool can create files for submission to the Synopsys BC tool. The BC allows for the synthesis of a wide variety of alternate architectures, each of which meet the original input behavioral description and input constraints. These various architectures are expressed as synthesizable RTL and are subsequently translated to component specific implementations through the Synopsys DC tool.

Requirement: Generation of the netlist representation.
Approach: The Synopsys DC and FPGA compiler take in RTL VHDL source from various design data sets including those referenced within the simulatable specification or stand-alone. Additionally, the combination of DC and BC facilitates the translation of behavioral VHDL that is provided through a graphical front-end tool, referenced by the simulatable specification, or stand-alone. The Synopsys DC also takes additional input to create a resulting design implementation. This input is in the form of a targeted component library (either ASIC or FPGA), designer library (basic building blocks), constraints for timing, area and power as well as a synthesis strategy. The DC also has capabilities for handling legacy gate-level netlists for re-optimization and re-targeting. The SimSpec/SDE tool flow, provided by this proposal, supports communicating that information from the simulatable specification or optional product implementation data set to the tools provided by Synopsys or other non-Synopsys-supported tools.

Requirement: Specification/modification by designer of the specific component types to be assumed for each component in the implementation.
Approach: The designer can specify alternative architectures as well as alternative targeted semiconductor libraries (both FPGA and ASIC). This proposal does not support board level re-mapping of components and therefore does not support specification or modification of components beyond that contained within a given IC (the whole IC).

Requirement: Interaction with the designer to create/modify the synthesis scripts used in the synthesis of each custom component.

Approach: The SDE is constructed in such a way that a baseline synthesis script is derived from data extracted from the implementation data set (or simulatable specification) as well as defaults for standard constraints. Users have access to these script modules for further modification and customization.

Requirement: The synthesis of the behavioral VHDL design for each custom component.

Approach: This capability is directly supported with the Synopsys BC.

Requirement: The interaction with VHDL simulation tools to be used to verify both the synthesizable VHDL design as well as the synthesized VHDL design for each custom component.

Approach: This interaction is supported by providing correct translation between the synthesizable (RTL) VHDL and synthesized (gate level) VHDL. Additionally, SDE is compatible with various verification environments and includes support for some components of verification (such as RTL regressions and static timing verification). It is, however, beyond the scope of this proposal to support a simulation framework or simulation strategy.

Table 6 shows in summary form how Synopsys' technical approach satisfies its requirements.

Table 6. BPR Requirements Summary

Requirement	Requirement Satisfied
Schematic view	Yes : View only
Mapping of signals and entities	Yes
Generation of synthesizable VHDL packages	Yes
Generation of the architecture for the design	Yes: Behavioral synthesis
Generation of the implementation netlist	Yes: Synthesis
Specification/modification by designer of specific component types	Yes: Limited to ASIC/FPGA libraries
Interaction with designer to create/modify synthesis scripts	Yes
Synthesis of behavioral VHDL design for each component	Yes
Interaction with simulation and verification tools	Yes: RTL regressions in SDE. Behavioral and RTL coordinate with verification tools.

5.2 DVTG Results

As can be seen from the implementation description discussed in section 3.2.2.2, the DVTG tool effort meets all of the objectives defined.

5.2.1 DVTG Requirements and Their Fulfillment

The following provides a brief description of each of the key requirements as well as how the requirement was satisfied for the DVTG tool.

Requirement: Generation of concrete test vectors.
Description: Demonstrate generation of concrete test vectors from abstract Rosetta functional requirements and test requirements representations. Specifically, demonstrate a flow from Rosetta requirements specifications and test requirements specifications through abstract test vectors to concrete test vectors compatible with an existing simulation system.
Approach: The DVTG tool accepts Rosetta descriptions written in the axiomatic style and automatically generates concrete test vectors in the WAVES format. System specifications are used to generate test scenarios. System requirements are used with test scenarios to generate abstract test vectors represented in Rosetta. Abstract test vectors are transformed into concrete test vectors for specific languages and tools. The methodology has been demonstrated for WAVES output, but is not limited to WAVES.

Requirement: Support representation of test requirements.
Description: Specifically, design and implement a Rosetta test requirements specification facet. The test requirements facet should be capable of representing test coverage information and specify constraints on values involved in tests.
Approach: Test requirements are specified in the following two ways:

- coverage requirements
- input signal requirements

Coverage requirements indicate ranges and step sizes for input values. They specify coverage required for correctness assurance. Input signal requirements define characteristics of input signals that require testing. They specify test requirements in a more traditional way, allowing the user to specify input signals directly rather than having DVTG generate them automatically.

Requirement: Support representation of test scenarios.
Description: Specifically, design and implement a Rosetta test scenario specification facet. The test scenario facet should be capable of representing abstract sets of tests to be performed on systems.
Approach: Test scenarios are specified using a special Rosetta facet definition. They specify different cases that assure requirements coverage.

Requirement:	Support representation of abstract test vectors.
Description:	Specifically, design and implement a Rosetta test vector specification facet. The test vector facet should be capable of representing specific tests independent from any specific simulation language. The test vector facet should minimally be capable of representing vectors of input values and associated output values and/or tests to determine correctness of output. The option of developing a single Rosetta test facet representing information from Rosetta test vector, test requirements, and test scenario facets, should be an option for implementation.
Approach:	Abstract test vectors are specified using a special Rosetta facet definition. They allow specification of driving and driven values in a system-neutral fashion. Abstract test vectors are converted to concrete vectors by vector generation tools.

Requirement:	Demonstrate generation of concrete test vectors from abstract test vectors.
Description:	Specifically, develop a mechanism for generating test vectors in a concrete language such as WAVES. Demonstrate simulation of VHDL model using generates WAVES test vectors.
Approach:	Both the SATCOM application and alarm clock examples demonstrate the generation of WAVES vectors from abstract vectors.

Requirement:	Support both white box and black box testing.
Description:	Specifically, address issues of test vector generation when internal structure is known and when internal structure is hidden.
Approach:	The SATCOM application and alarm clock examples both employ black box testing. The alarm clock example also employs white box testing by examining requirements during vector generation.

Requirement:	Integrate test vector generation into the Mentor Graphics Renoir environment.
Description:	Use Renoir to generate test benches and manage the test generation process. Determine feasibility of using Renoir to graphically manage Renoir specifications, generate WAVES test benches, WAVES test vectors, and accompanying VHDL models.
Approach:	Our commercialization partner, EDAptive Computing, has integrated the DVTG system into the Renoir environment. They have demonstrated the capability and are providing a beta release for potential customers.

Table 7 shows in summary form how the technical approach for DVTG satisfies the requirements.

Table 7. DVTG Requirements Summary

Requirement	Requirement Satisfied
Demonstrate generation of concrete test vectors	Yes
Support representation of test requirements	Yes
Support representation of test scenarios	Yes
Support representation of abstract test vectors	Yes
Demonstrate generation of concrete test vectors from abstract test vectors	Yes, for VHDL only
Support both white box and black box testing	Yes
Integrate test vector generation into the Mentor Graphics Renoir environment	Yes

5.3 Demonstration Results

The following were the key objectives for the demonstrations, as detailed in section 4.2:

1. Developing a "releasable" design, allowing for the distribution of the demonstration
2. Developing a demonstration that includes a military application
3. Developing a design scope which is of significant size but does not require major development, since our time is limited and our budget is constrained
4. Developing a demonstration that uses a mix of behavioral VHDL and RTL VHDL.

Table 8 provides a summary of the objectives and which primary demonstration satisfies the objective.

Table 8. Demonstration Objectives Summary

Objective	Demonstration
Releasable design	Behavioral alarm clock
Military application	SATCOM application
Significant scope	SATCOM application
VHDL mix	Behavioral alarm clock

Following is a textual description of how these key requirements are met:

- Releasable Design: The behavioral alarm clock demonstration best satisfies this objective because it provides an end-to-end illustration of the methodology and tools. It is also not proprietary in nature (as opposed to the SATCOM application demonstration) and all of its facets are available for inspection. These include the following:
  - Requirements facet

- Test facet
- Operational facet

It is also simple enough that its function and respective facets should be relatively easy to understand.

- Military Application: The SATCOM application demonstration best satisfies this objective because of its potential use in military applications such as Comanche and JSF. Unfortunately, due to its proprietary nature, only the requirements facet and test facet will be available for inspection.

- Significant Scope: The SATCOM application demonstration best satisfies this objective due to its robustness. One advantage of this demonstration target is that it allowed for adding functionality (and waveforms) incrementally throughout the development effort.

- VHDL Mix: The behavioral alarm clock demonstration best satisfies this objective because it provides a mix of both behavioral VHDL and RTL VHDL. Another demonstration that satisfied this objective was the behavioral Core demonstration.

6. SUMMARY/LESSONS LEARNED

In summary, all goals set for the program have been met. The CEENSS methodology has been extended to incorporate the reengineering design flow and tools were developed to facilitate that flow. Demonstrations were also conducted to test the methodology and tools as well as illustrate their capabilities. Each of these are described in following sections. Prior to that, however, it may be beneficial to illustrate how TRW's POMT effort fits into the overall initiative.

The POMT effort described in this report is only one part of a manufacturing technology initiative. As described in Anthony Bumbalough's white paper:

> *This initiative currently consists of eight programs covering three key areas of work.*
>
> 1) *Parts Obsolescence Management and Re-engineering Tools*
> 2) *The Application of Commercially Manufactured Electronics (ACME)*
> 3) *Pilot Demonstration Programs*
>
> *The initiative's main technology focuses are mixed signal electronics, ASICs, Physics of Failure validation with commercial field return data, and standardized information exchange.*

The TRW POMT effort is part of the reengineering tools effort and continues previous work from a manufacturing technology program (the CEENSS program) on system top-down design and simulatable specifications.

The goals of the pilot demonstration programs as described by Anthony Bumbalough in his white paper are as follows:

> *… to demonstrate technology insertion to systems, and to develop and document the obsolescence management business case. The initiative uses pilot demonstration programs to insure the successful demonstration and transition of the best business practices, tools and technology developed by the initiative. The objectives are to insure reliable application of commercial electronics in military systems, while documenting the cost avoidance of COTS and corporate approach to managing obsolescence.*

Relative to TRW's POMT effort, the pilot demonstration programs were to evaluate the tools developed and, if applicable, demonstrate their capability. In retrospect, it would have been beneficial to start the pilot demonstration programs earlier relative to TRW's POMT effort. Much of the methodology was already defined and the requirements for the individual tool efforts set prior to the start of the pilot demonstration programs. Neither the DVTG nor BPR tools were influenced significantly by the programs tasked with evaluating and potentially demonstrating their capabilities.

6.1 VSPEC to Rosetta

During this effort, a decision was made to move from VSPEC as a source language to the emerging Rosetta systems-level design language. It became clear that basing a commercial system on VSPEC would be difficult for reasons concerning standardization and movement within the CAD community. The decision was made to move the DVTG aspects of this effort to the Rosetta language environment. It was felt that Rosetta had more commercial pull and represented a potentially lucrative basis for test vector generation.

Although long-term potential is being met by the Rosetta effort, in the short term, significant risks were involved. The Rosetta language is not yet a standard, and tools are university beta quality. However, team members are intimately involved in the Rosetta effort and have required tool expertise to make the ultimate result workable. Significant new interest in Rosetta has emerged, making the move to Rosetta an excellent choice in retrospect.

Growing pains with respect to Rosetta usage can be viewed as both technical and social. Not surprisingly, the technical issues have proven far easier to overcome than social issues. Moving the DVTG algorithms to the Rosetta parser has proven straightforward due to design decisions made during parser development. The Rosetta object model and parser proved easier to work with than the original VSPEC tool set. Although the language specification has been fluid, none of the changes have dramatically affected the DVTG effort. In most cases, language changes have made the tool more powerful and easier to develop. Specific examples include the MATLAB™ library files used for test requirement specification and the ability to express vectors and requirements directly in Rosetta. These features dramatically simplified the vector generation process and would not have been doable with the original VSPEC language without significant effort.

Social issues have been more difficult, but not fatal to the effort. Lack of a language standard and working with an emerging language introduce risk into the process. Social risks include adoption of the language by the community and the willingness of engineers to write specifications in a new language. Current efforts in systems level design languages suggest two alternate approaches:

- extend existing languages
- develop new languages

Rosetta is an example of the latter situation. A clear winner has not been established; however, it is most likely that a mixture of the two approaches will succeed.

6.2 Methodology Summary

The POMT methodology is capable of handling legacy design, initial product design, and reengineering an existing design as described in section 3.1. The primary requirement to realize the benefits of the methodology however is that a simulatable specification be developed for the design. This starts with the development of a Rosetta requirements model (also known as the

requirements facet). This approach appears to be better suited for an initial product design rather than recovering a legacy design because it effectively starts from scratch and makes it easier to determine the actual requirements for the design as opposed to attempting to reverse engineer the requirements from an implementation.

In the case of recovering a legacy design, a decision needs to be made whether or not the simulatable specification approach is the most cost effective (please see section 3.1.1). As described in section 3.1.1, there are other tools that may be used in the place of those in TRW's POMT methodology or that can be used to support the methodology.

Initially, there is additional work involved in the development of the requirements model. It is a more intensive effort than the standard approach of developing requirements in the form of a paper specification, but benefits can be realized the first time the design is implemented with a savings in the integration and test stages (please see section 3.1.4.2). Significant benefits are realized, however, when it comes time to reengineer a design that implements this methodology.

The methodology also supports TRW's plan to adopt a P^4I strategy that integrates elements of design, manufacturing and support. This strategy preempts obsolete parts problems through preplanned periodic redesigns of the electronics, the recurring costs of which are recovered through savings in production and support costs. In addition, the periodic incorporation of new technology with higher processing densities allows for performance enhancements and savings in weight and cost. Contributing significantly toward incentive to adopt this strategy is the adoption of commercial business practices and the move towards CLS.

6.3 Tool Summary

In summary, all of the goals and requirements for the POMT tool efforts have been met, as detailed in sections 5.1 and 5.2.

One of the primary objectives of TRW's POMT effort was to incorporate tools from commercial tool vendors into the DoD environment. This is effectively a two-way street. In one direction, DoD needs to be aware of what factors go into a commercially viable tool. If DoD requires a capability that only the DoD will use, it is unlikely that such a capability will find its way into a mainstream shrink-wrapped tool that will be supported by the vendor. As discussed earlier, DoD does not have enough market share to drive the tools. The DoD represents such a small percentage of the customer base, that it would not make good business sense for a vendor to modify tools with capabilities that support only the DoD environment.

In the other direction, tool vendors need to be educated about what is different in regard to the DoD environment relative to the commercial environment. With a clear understanding of the needs for both the commercial and DoD environments, developing a tool that addresses both is a much simpler task. The ultimate goal is to develop a tool that can support both environments

without sacrificing capability. Even though there are some key differences between the two environments (such as life cycle), both environments are similar at their most basic levels.

Benefits of having commercially available tools that support the DoD environment are substantial. There are currently significant gaps in the design process not being supported by commercial tools. Often, custom tools need to be developed at great expense. Also, if commercial standards are adopted, information flow through the tools becomes much simpler.

Relative to the tools developed on TRW's POMT effort, it would have been desirable if the Synopsys effort resulted in a commercially available shrink-wrapped product. The end result of the effort was a tool developed out of their Professional Services Group that requires customization for the user's specific tool environment and requires additional support with new releases of the tools within that environment. The Synopsys effort did however prove that a fairly seamless flow from development of a simulatable specification through design synthesis is possible. As described above, the primary obstacle for tool development and support from the large EDA houses is the lack of a business case for investment into the tools. As SLDL's become more adopted, the business case will result in the larger CAD vendors (such as Synopsys) supporting shrink-wrap products that provide the same capability as BPR.

6.3.1 BPR Summary

The purpose of the BPR tool developed by Synopsys for TRW is to support the technical community in the area of design reuse, design reengineering and parts obsolescence management tools. Synopsys has developed the appropriate tools, flow and overall environment to allow for behavioral/RTL design and architectural re-targeting of designs that have been extracted from legacy design documentation and information by some other process.

The SimSpec/SDE embodiment of the POMT program provides a methodology whereby an implementation-independent representation of a design can be taken as input. Architectural trade-offs can be explored and the implementation-specific outputs generated to enable a physical realization of the design in a wide variety of targeted technologies.

The SDE demonstration vehicle has been developed with capabilities to interface with the simulatable specification through an external simulatable specification parser (provided by the University of Cincinnati's DVTG effort) and the SimSpec2SDESetup tool (provided by Synopsys). In addition, Synopsys has provided an example design and documentation suite that will demonstrate the design environment flow through the SimSpec2SDESetup Tool, SDE and the Synopsys standard tools.

Synopsys has proven its ability to deliver this capability to the design flow required for supporting ICs re-targeting of both architecture and technology. Capabilities in behavioral synthesis provide a mechanism for moving a design object through its various possible implementations over a 20 year period. Synopsys' DC is a proven tool that can realize an implementation with timing, area and power constraints as inputs to the synthesis process. In

addition, Synopsys provides other tools in the area of test bench creation and automated test generation. In addition to offering seamless integration of the Synopsys tools, the SimSpec/SDE environment provides an open means to integrate other EDA design tools and enables the integration of future tools and capabilities as the state of the art progresses.

Synopsys Professional Services Group has a history of working with many variations of inputs to its various leading technology design tools and solving this type of integration challenge for customers in completing customized tool flows. Synopsys has conducted and continues to conduct research into a wide variety of subjects related to high level design, system-on-chip design, design reuse, intellectual property component repository systems, test bench automation, behavioral synthesis, logic synthesis, design creation, and design verification. Synopsys team members are often authors of trade studies and technical white papers on capabilities in these domains. Such efforts ensure that Synopsys tools and design environments will keep pace with and play a key role in defining the chip design methodologies of the future.

For further detail regarding the capabilities and benefits of the BPR effort, please refer to section 3.2.1 (section 3.2.1.2.4 in particular).

6.3.2 DVTG Summary

The purpose of the DVTG tool developed by The University of Kansas, University of Cincinnati, and EDAptive Computing is to support automated testing of systems. This capability supports the legacy systems support objective in two ways as follows:

- engineers can automatically test legacy components against specifications
- engineers can automatically test new components against design specifications.

The first capability allows the design engineer to determine if the specification generated from a legacy component is in fact a faithful description of the component. Such information is vital in support of redesign efforts when replacing components. The second, more traditional capability allows the design engineer to determine if a design result is in fact meeting end requirements.

Although many tools exist for generating test vectors, DVTG is unique in its capability for automatically analyzing specifications and generating test cases that provide guaranteed requirements coverage. Because vectors are generated directly from requirements rather than an operational VHDL model written from requirements, the DVTG algorithm can assure coverage. Thus, following testing, the design engineer can proceed with confidence that requirements have been covered during the testing process. Furthermore, when requirements change, the design engineer can easily generate new test vectors that are compliant with the new requirements.

The DVTG prototype was designed in a manner independent from existing simulation languages and tools. Rosetta system requirements are transformed into collections of test scenarios that describe various situations that must be evaluated to assure requirements coverage. Test requirements are then used to generate specific test cases from the test scenarios generated from functional requirements. The resulting test cases, called abstract test vectors, are

expressed in a manner independent from any specific simulation engine. Transformation tools then generate test vectors for specific test systems and languages from the abstract vectors.

EDAptive Computing developed concrete demonstrations of DVTG in support for an eventual commercial offering. A plug-in for the Renoir graphical environment was developed and support for generating WAVES vectors was implemented using the DVTG framework. The WAVES generation capability converts abstract test vectors into WAVES vectors suitable for use in VHDL. The resulting outputs were tested as part of the SATCOM application demonstration described in section 4.3.5. This demonstration is evidence of the validity of the general approach. The Renoir plug-in integrates DVTG capabilities into the environment. Renoir demonstrates the effectiveness of the approach when included in a tool integration environment. Section 10 describes other related tool efforts by EDAptive.

6.3.2.1 Rosetta Future Activities

Work on Rosetta continues and is growing throughout the world. The VHDL International and Open Verilog International organizations have merged into a single CAD standards organization called Accellera. The systems level design language effort has moved from VI to Accellera and continues to be actively pursued. Standardization processes will begin within Accellera in late 2001 or 2002. It is anticipated that a commercial demonstration with Texas Instruments will begin in the second quarter of 2001, sponsored by Accellera. This demonstration will provide the catalyst for standardization processes. A Rosetta book is being written, and the authors are under contract to deliver a published book by the 2002 Design Automation Conference (DAC).

Tool development continues with The University of Kansas, Adelaide University, AverStar, and EDAptive performing research and developing commercial Rosetta tools. The University of Kansas continues to develop front-end parsing capabilities as well as component retrieval and test vector generation capabilities. The University of Kansas has started prototyping an evaluation system for Rosetta specifications and are engaged with AverStar and EDAptive in commercial tool development. Further commercialization is anticipated through The University of Kansas Information and Telecommunications Technology Center following development of research prototypes.

Adelaide University has recently begun an effort to develop Rosetta simulation tools. Dr. Peter Ashenden and Dr. Robert Esser are leading a 3 year effort to develop Rosetta simulation capabilities.

6.4 Demonstration Summary

In summary, the demonstrations met all of the objectives detailed in section 4.2. No single demonstration, within scope and budget, was capable of demonstrating a significant portion of the methodology and tools. Multiple limited scope examples were developed to test different aspects of the methodology and tools. The two primary demonstrations were the behavioral alarm clock and the SATCOM application.

From a metrics point of view, the behavioral alarm clock was not robust enough to provide a reasonable picture of the expected benefits. Its primary purpose was to illustrate the capabilities of the individual tools as well as how the tools tie into the methodology. It also provides an example that can be used for educational purposes. It is an excellent starting point for understanding the various facets involved and how those facets are related.

The SATCOM application demonstration provided a much clearer picture of the benefits associated with the tools and methodology. It was also used to test the methodology and the tools as well as to illustrate the capabilities of the methodology and tools. Initially, the demonstration was used during the development of the DVTG tool for debug and test purposes. The demonstration was also used to illustrate the capabilities of the individual tools as well as how the tools tie in to the methodology.

For the SATCOM application, the benefits realized were in line with those projected in Table 2 in section 3.1.4.2. For the POMT effort, the baseline is the CEENSS methodology, thereby putting the focus on the benefits realized in the synthesize product implementation and develop reference test steps.

6.4.1 SATCOM Application - BPR Lessons Learned

The use of the BPR tool did reduce the effort involved in synthesizing the implementation, although the benefit was slightly degraded by the use of a third party tool. Section 4.4.1.3 details the deviation from the methodology by the use of this tool.

One primary benefit that was realized in the synthesize product implementation step was that the design of the implementation was easier because the requirements facet could be used as a starting point for the implementation. Another benefit was that the information flow through the tools was more automated with the introduction of SDE and its ability to extract synthesis constraints from the requirements facet. One difficulty in this step, however, was that we were using an NT workstation for most of the implementation development, and then used a UNIX workstation for the actual synthesis effort. This resulted in extra effort of porting all of the necessary information between the two platforms.

6.4.2 SATCOM Application - DVTG Lessons Learned

The benefit in the develop reference test step was even more obvious with the use of the DVTG for partially automated test vector generation. There were some growing pains, however.

The greatest potential benefit from the proposed design flow lies in the automated generation of test vectors from the requirements. However, the SLDL was not initially well suited to the task of generating test vectors from system-level performance requirements. For example, we have a system requirement to detect a unique word a certain percentage of the time given an input signal-to-noise ratio. This requirement is not an input to the system, but rather a description of

an input waveform property. In order to handle this requirement, the test facet was expanded to include a waveform generation component. The DVTG tool was also not initially geared to handle a sequential input, a must for the type of application we are demonstrating. Instead, DVTG was designed to produce single values that were coupled with a state internal to the device under test. Additionally, no capability existed to test some of the outputs. Another of the performance requirements was to receive data with a bit error rate less than some value. This was another case of the requirement not being a definition of the output signal, but rather a property of the output waveform. In order to test this requirement, additional functionality was proposed for the implementation.

6.4.3 SATCOM Application - Other Lessons Learned

There was a tendency to want to embed functionality into the requirements facet. It is an engineer's nature to think about how a design will be implemented as opposed to the black box requirements. When writing the requirements facet, it is important to describe required function without describing an implementation of that function.

It is anticipated that the benefits realized with this methodology and tools will be magnified when a reengineering effort is required for the SATCOM application. As discussed previously, the primary benefit with this methodology is when a reengineering activity takes place. By designing at a higher level of abstraction and partially automating the test vector generation process, there is a significant reduction in the overall reengineering effort. Table 3 in section 3.1.4.4 illustrates anticipated benefits with this activity.

7. CONCLUSION

Throughout TRW's POMT effort, it became apparent that a formal specification of requirements has many benefits. It is also apparent that both the tools and methodology are integral to the design flow. A great methodology without tools that support it is not very useful. Likewise, there may be great tools that solve a critical piece of a design challenge whose benefit is diminished if it does not fit into the overall design methodology.

The two tools that were developed on this effort are the DVTG and BPR. Each attacks a critical portion of the methodology. The focus of the BPR tool was to provide a mechanism to incorporate constraints specified in the simulatable specification into the synthesis environment. The BPR tool also incorporated Synopsys' BC in the tool flow to allow for designing at a higher level of abstraction than standard RTL VHDL. The methodology has taken this one step further with the incorporation of Rosetta (an SLDL) for describing the requirements model. Rosetta allows for describing the design in a non implementation-specific manner. The user can describe the requirements of the design as opposed to how it is implemented, the what instead of the how. It also allows for multiple domains to be described, effectively allowing for a higher level or system view of the design. Rosetta is in its infancy, but work has already begun to develop a Rosetta simulator. We expect that direct synthesis from a Rosetta model will be available in the future.

The other tool developed is the DVTG tool. The focus of this tool was to partially automate test vector generation. These test vectors are generated from the requirements specification as opposed to the implementation, which significantly enhances the reuse capability of a design. The DVTG was developed to accommodate the focus of TRW's POMT effort, but it also has room to grow. We anticipate, based on interest, that continued commercial development of DVTG and related standards and tools will occur. This includes the SLDL standard, the Rosetta language, and supporting simulation and analysis tools. This continued development will result in a more robust DVTG capability in the future. We believe this capability will result in significantly reducing the effort involved in validating an implementation against its requirements. This capability is anticipated to be a significant step forward in the reengineering of future product designs.

8. GLOSSARY

AFRL	Air Force Research Laboratory
AGC	automatic gain control
ASCII	American National Standard Code for Information Interchange
ASIC	application-specific integrated circuit
ATM	asynchronous transfer mode
ATVG	automated test vector generation
BC	Behavioral Compiler (Synopsys Tool)
BPR	Behavioral Product Reengineering
CAD	computer-aided design
CEENSS	Continuous Electronic ENhancements using Simulatable Specifications
CLS	contractor logistics support
CNI	communications, navigation and identification
COTS	commercial off-the-shelf
DAC	Design Automation Conference
DC	Design Compiler (Synopsys Tool)
DF	design file
DMS	diminished manufacturing source
DoD	Department of Defense
DSP	digital signal processor
DVTG	Design Verification Test Generator
ECSI	European Chips and Systems Initiative
EDA	electronic design automation
EMD	engineering manufacturing development
EMI	electromagnetic interference
EPO	electronic parts obsolescence
EPOI	Electronic Parts Obsolescence Initiative
F^3I	form, fit, function and interface
FPGA	field programmable gate array
GL	gate level
HDL	hardware description language
I/O	input/output
IC	integrated circuit
IP	intellectual property
JSF	Joint Strike Fighter
LRU	line replaceable unit
MCM	multi-chip module
MEMS	microelectromechanical systems
OPP	out of production parts
P^4I	periodic preplanned product improvement
POMT	Parts Obsolescence Management Tools
PVS	Prototype Verification System
RF	radio frequency
RTL	register transfer level
SATCOM	satellite communications

SBIR	Small Business Innovation Research
SDE	Synopsys Design Environment
SLDL	system level description language
TCF	top constraints file
TDMA	time division multiple access
VHDL	VHSIC hardware description language (IEEE Std. 1076)
VHDL-AMS	VHDL – Analog Mixed Signal (IEEE Std. 1076.1)
VHSIC	very high speed integrated circuit
VI	VHDL International
VSIA	Virtual Socket Interface Alliance
VSPEC	VHDL specification
WAVES	waveform and vector exchange (IEEE Std. 1029)

9. APPENDIX A - ROSETTA

Rosetta is an emerging SLDL, useful for describing various aspects of computing systems. Technically, Rosetta supports simultaneous specification of heterogeneous systems models. Using the facet concept, Rosetta supports defining and composing models to define components and systems. It is domain and technology neutral, allowing specification of software, hardware, and mechanical systems. Politically, Rosetta is well positioned to become a future specification standard. Rosetta efforts are sponsored and supported by the U.S. Air Force Research Labs (AFRL), VHDL International (VI), the European Chips and Systems Initiative (ECSI) and is being evaluated by the Virtual Socket Interface Alliance (VSIA). Because of its technology independence and emergence as a potential standard, Rosetta is an excellent choice for system specification.

9.1 Rosetta Specification

Rosetta's basic specification unit is a design facet or simply a facet. Each facet describes one view or model of a system or system component. Figure 32 shows two such facets, describing functional requirements and performance constraints for a small system. The structure of the specifications follows a familiar form, providing parameterization, communication, data item declaration, and a mechanism for defining facet properties. The distinction between a Rosetta specification and specifications written in a traditional specification language is the explicit inclusion of a domain specified as a part of the begin declaration. Each facet is based on a domain specification that defines a vocabulary for that facet. The functional requirements specification, sort-req, uses a domain for specifying axiomatic requirements, while the performance requirements specification, sort-const, uses a constraint specification domain that defines power, timing, and other performance constraint items. Thus, the designer is not forced to use "least common denominator" specification semantics.

Complex, multifaceted systems are modeled by defining and composing facets from various design domains. For example, when modeling complex digital systems, functional correctness and performance constraints are issues that must be considered when assessing correctness. Using the Rosetta system, the designer specifies a performance constraint facet and a functional correctness facet. The domain specific facets are combined using conjunction to form a new facet that describes the desired system. This new facet has both sets of properties simultaneously. Figure 32 shows an example Rosetta fragment that specifies a functional requirements facet, a performance constraint facet and the use of conjunction to define an overall component. It is a simplistic example of the approach in which functional correctness of a sorting algorithm is specified by the facet sort-req and power consumption constraint specified by facet sort-const. The conjunction, facet sort, is a specification in which both specifications simultaneously hold.

```
facet sort-req(i::in sequence(T); o::out sequence(T)) is
begin continuous
  pre: true;
  post: permute(o',i) and ordered(o');
end sort-req;

Sort = sort-req and sort-const;
```

```
facet sort-const
  power::real;
begin constraints
  p1: power <= 5mW;
end sort-const
```

Figure 32. Example Rosetta Fragment

In addition to specifying individual components, Rosetta provides mechanisms for specifying system structure using techniques similar to those employed by VHDL. Components are instantiated and interconnected to represent systems that use aggregation. Like facets for a single component, facets representing components in a structural description may be heterogeneous in nature.

9.2 Domains

Each Rosetta domain provides a vocabulary for defining specifications in a particular design domain. When writing a design facet, the Rosetta user selects a domain and then extends the domain by writing a specification for the current problem. The example sort-req in Figure 32 uses the state-based domain as the basis of specification. This domain provides the semantics for referencing an item at the current or the next state. Thus, o' has meaning in this domain. Alternate specification domains provide a means for defining discrete time, continuous time, and performance constraint specifications. The example, sort-const, uses the domain constraints to define a single performance constraint for the component. This domain defines properties such as heat dissipation, power consumption, and timing constraints.

Existing Rosetta domains address specification of functional requirements and performance constraints in design problems related to systems-on-a-chip processes. Specification domains currently exist for the following:

- general purpose mathematical specification
- state-based specification
- finite state systems
- synchronous-reactive systems
- discrete time systems
- continuous time systems.

In addition to these functional domains, the constraints domain exists for defining performance constraints, and a mechanical domain is being developed to describe mechanical systems requirements. The constraints domain is being developed and enhanced in conjunction with the Open Verilog International / VHDL International Design Constraints Working Group to assure industrial acceptance and applicability. The mechanical systems domain is being developed in conjunction with the Air Force Materials Directorate to assure applicability to DoD systems.

9.3 Domain Interactions

Unlike traditional specification languages in which all specifications share a common semantics, Rosetta provides a framework for defining and composing specifications with multiple underlying semantic models. Such a framework is necessary to allow domain experts to design, using abstractions common to their engineering domain while supporting domain integration. This framework does not rely on a shared semantics, but on a mechanism for specifying domain interactions. Thus, users write Rosetta specifications using semantics and language constructs suitable for their specific domain. When specifications are composed, the interaction framework is used to analyze consistency of the composition.

Domain interactions are defined in Rosetta using a domain interaction or simply interaction. Such definitions use Rosetta's reflective specification capabilities to define when and how interactions occur. In many cases, such interactions are easily modeled and evaluated. One such case is the interaction between a safety specification and a state-based specification. We know that any safety property, typically expressed in a time-invariant manner, applies to all system states. Other interactions are far more complex and cannot be completely characterized. Even in these cases, it is possible to coarsely specify interactions between domains. The key is to define interactions that cause specifications to interact only when appropriate rather than provide complete theories that become mathematically intractable.

10. APPENDIX B - EDAPTIVE RELATED TOOL EFFORTS

10.1 Introduction

In addition to the POMT effort, EDAptive Computing, Inc. (EDAptive) and the University of Kansas are jointly working on several other related design automation tool research and development efforts that will collectively solve the electronics parts obsolescence (EPO) problem. Specifically, the University of Kansas is conducting research in the areas of system level design languages, specification-based component retrieval and automated test vector generation whereas EDAptive is serving as the technology transition partner, transforming research technologies and prototypes from the University of Kansas into real-world solutions under the sponsorship of the U.S. Air Force, DARPA, US Navy, and NASA through several Small Business Innovation Research (SBIR) projects. To provide a good understanding of ongoing and possible future-related tool developments, an overview of the EDAptive EPO methodology is provided, and then each of the tool components of this methodology is described.

10.2 Overview

EDAptive EPO methodology utilizes four core technologies, namely SLDL to specify system requirements, a system design editor to capture system specifications intuitively, DVTG to automate test generation and rapid component retrieval to enable retrieval of components that match requirements. Further, EDAptive EPO methodology has been primarily designed for obsolete electronic board replacement, although could be easily customized for other electronic systems. Figure 33 shows an overview of the EDAptive EPO methodology and the relationship between the core technologies it employs.

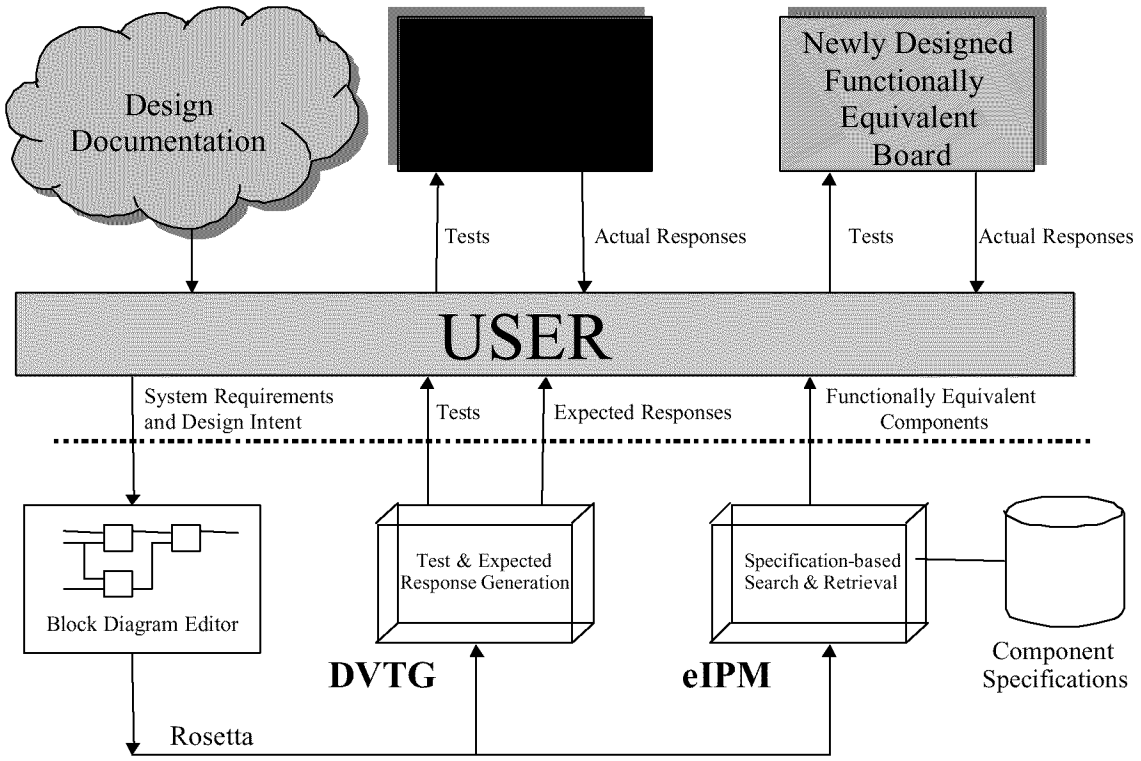

Figure 33. EDAptive EPO Methodology

Under EDAptive's EPO methodology, a user will create a Rosetta (a SLDL) specification of the obsolete board using available design documentation for the board, such as schematics, test data, and requirements. With Rosetta, all required electronic systems, namely general ICs and ASICs, analog circuits, digital circuits - both logic and signal processing functions, and hybrid circuits can be specified. EDAptive's system design editor, namely Syscape™ , can be utilized to capture a Rosetta specification more intuitively. Once created, a user will employ the DVTG-based tool, namely EDAptive VectorGen™, to generate tests and corresponding expected responses from the Rosetta specification. The user will then validate the Rosetta specification by testing the actual, obsolete board with generated tests and comparing its responses with the DVTG-generated expected responses. The user will continue to refine the Rosetta specification until the test of the actual obsolete board yields the same response as the DVTG-generated expected response. Then, the user will employ the EDAptive eIPM™ tool suite to search for and retrieve components from a database of component specifications that partially or fully match the Rosetta specification. Using the found components, the user will construct a functionally equivalent board. Following construction of the functionally equivalent board, the user will test the newly constructed board with DVTG-generated tests to validate the board design. The user will repeat the last two steps until the newly constructed board's response matches the DVTG-generated expected response. Figure 34 shows all of the steps required to construct a new, functionally equivalent system using a flowchart.

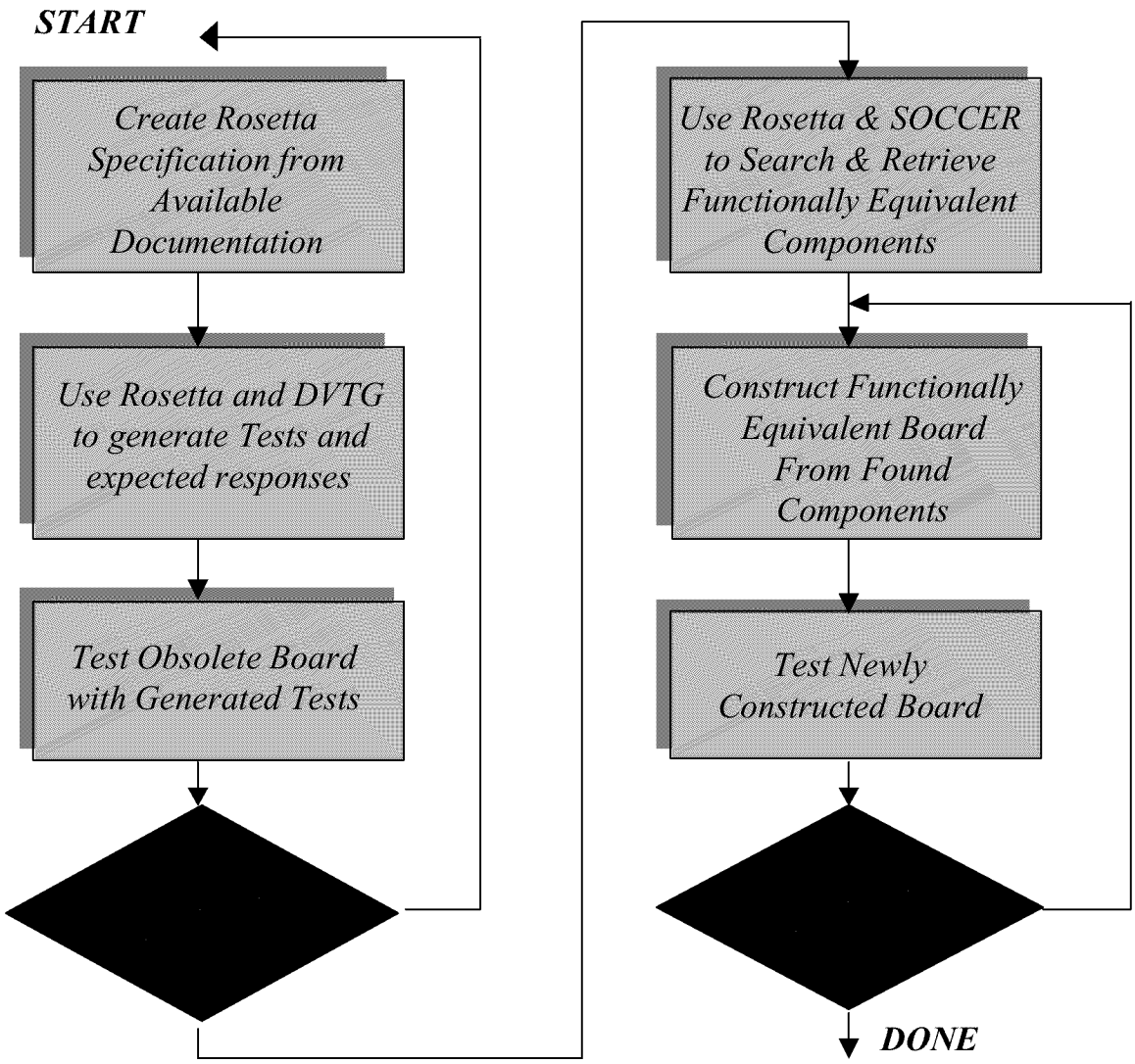

Figure 34. Steps in EDAptive EPO Methodology

Next, a brief overview of each tool component is provided, as shown in Figure 34, namely Syscape™, VectorGen™ and eIPM™. These tools have resulted from ongoing research and development efforts, and EDAptive continues to explore several enhancements to these tools in collaboration with the University of Kansas. More information about EDAptive's tool offerings is available on its website at www.edaptive.com.

10.3 Syscape™

10.3.1 The Challenge

Syscape™ is a solution for commercial enterprises and government agencies experiencing difficulty in fielding mixed technology systems of growing complexity due to lack of design

automation tools. The difficulty in fielding complex, heterogeneous systems is compounded by the shrinking scientific, technical, and engineering workforce. Mixed technology systems and the shrinking knowledge workforce are the drivers that motivate our Syscape™.

10.3.2 The Concept

The primary goal of Syscape™ is to provide a design environment in which designers of mixed technology systems can graphically capture their designs in multiple forms and varying levels of abstractions and further customize the design environment for easy use of their domain tools. To accomplish this goal, EDAptive is developing a unique product by researching, developing, adapting, and integrating the following functionality:

- Hierarchical block diagram editor with an ability to associate user-defined multiple views with each block in the design
- Support for user-defined plug-ins to process system design information contained in the block diagram
- Support for design reuse through archival of designs in a library
- Plug-ins to support use of the design environment for capturing requirements in Rosetta and automatically generate requirements-level tests using EDAptive VectorGen™.

10.3.3 The Benefits

- Enables graphical capture of mixed technology system designs
- Permits capture of heterogeneous designs through association of user-defined views with elements of system design
- Permits customization of design environment with user-defined plug-ins
- Supports design reuse through archival of designs in libraries
- Plug-in to capture system specifications in Rosetta
- Plug-in to automatically generate tests from Rosetta specifications with EDAptive VectorGen™.

10.3.4 Key Features

- Hierarchical block diagram editor
- Ability to associate multiple views with each block in the design
- Support for user-defined plug-ins
- Platform-independent design
- Plug-in for generation of structural Rosetta
- Plug-in for use of EDAptive VectorGen™.

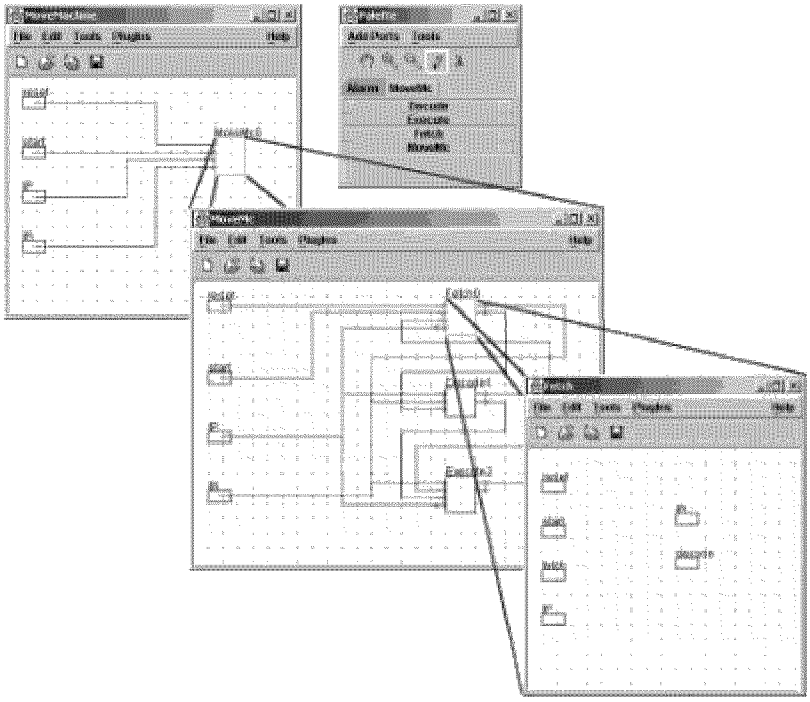

Figure 35. Syscape Design Environment

10.4 VectorGen™

10.4.1 The Challenge

Fifty to seventy percent of the total design cycle time is spent in verification, and more than fifty percent of the verification time is spent in modeling the environment and developing tests. In addition, paper-based requirements development and tracking leads to mistakes that are found during test and integration, when most expensive. Such mistakes account for over fifty percent of development cost and increase integration risk, costs, and delays. Reduction of verification time through automated test generation and use of formal specification is the driver that motivates VectorGen™.

10.4.2 The Concept

The primary goal of VectorGen™ is to automate test vector generation from specifications. This tool accepts the following as input:

- a requirements level system description
- test requirements, written in the Rosetta specification language.

The tool generates a set of generic test vectors that are useful for assessing the correctness of a detailed design with respect to the requirements specification. These generic test vectors are

transformed into vectors for a specific modeling language and environment. The current implementation addresses generation of WAVES test vectors suitable for evaluating a potential design described using traditional VHDL.

10.4.3 The Benefits

- Provides automated test generation capability
- Implements specification-based testing
- Enables selection of relevant test cases
- Provides generic test vectors
- Supports concrete test vector generation capability.

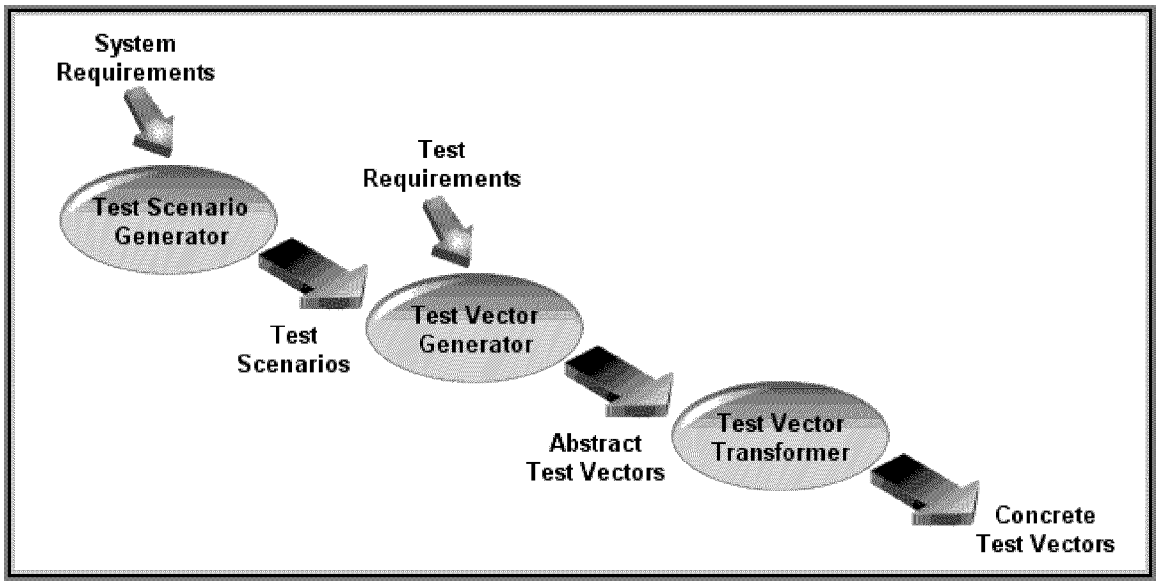

Figure 36. The VectorGen™ Information Flow

10.4.4 Key Features

- Platform-Independent Design
- Works for Rosetta Behavioral specifications
- Generates WAVES test vectors
- Allows Test Requirements Specification
- Easily integrates as plug-in to Design environments

10.5 eIPM™

10.5.1 The Challenge

eIPM™ is a solution for commercial enterprises and government agencies that are experiencing difficulty in fielding systems of growing complexity. The difficulty in fielding complex systems is compounded by the shrinking scientific, technical, and engineering workforce. Growing system complexity and the shrinking knowledge workforce are the drivers that motivate eIPM™.

10.5.2 The Concept

The primary goal of eIPM™ is to enable reuse of IP through accurate search and retrieval of IP. To accomplish this goal, EDAptive is developing a unique product by researching, developing, adapting, and integrating the following technologies:

- Formal specification language to capture the requirements and intended functions of desired system
- Multi-tiered, specification-based and parameter-based search and retrieval tool suite that accurately and rapidly finds IP that either fully or partially matches the system specification
- New databases of digital signal processor (DSP) components, analog components, MEMS reduced-order models, and ATM components for search and retrieval with EDAptive's tool suite.

10.5.3 The Benefits

- Enables design reuse
- Complete IP management suite
- Facilitates collaborative design
- Enables distributed design
- Enables correct-by-construction systems
- Supports specification-based system synthesis
- Supports architecture-based design

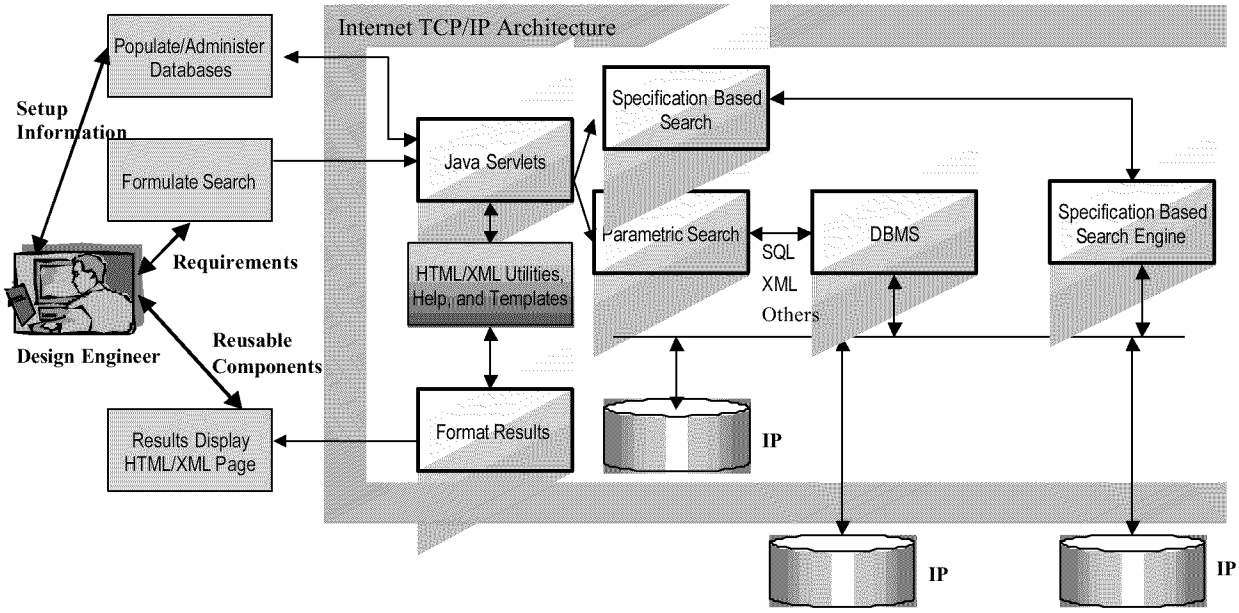

Figure 37. eIPM™ Architecture

10.5.4 Key Features

- Browser-based graphical user interface
- Parametric and specification-based search and retrieval
- Platform-independent design
- Easily integrates with third party tool environments to provide reuse support
- Search selected heterogeneous and distributed databases
- Works with most relational databases
- Fast and accurate search
- Ability to save searches between sessions
- Secure authentication system
- Customized viewing of search results
- Customized display preferences.

11. APPENDIX C – ALARM CLOCK DEMONSTRATION

11.1 Introduction

This appendix provides an overview of the behavioral alarm clock demonstration. For additional information, please see the Rosetta web page (www.sldl.org). Subsections provide both Rosetta and VHDL example code as well. Figure 38 provides an overview of the inputs and outputs of the alarm clock.

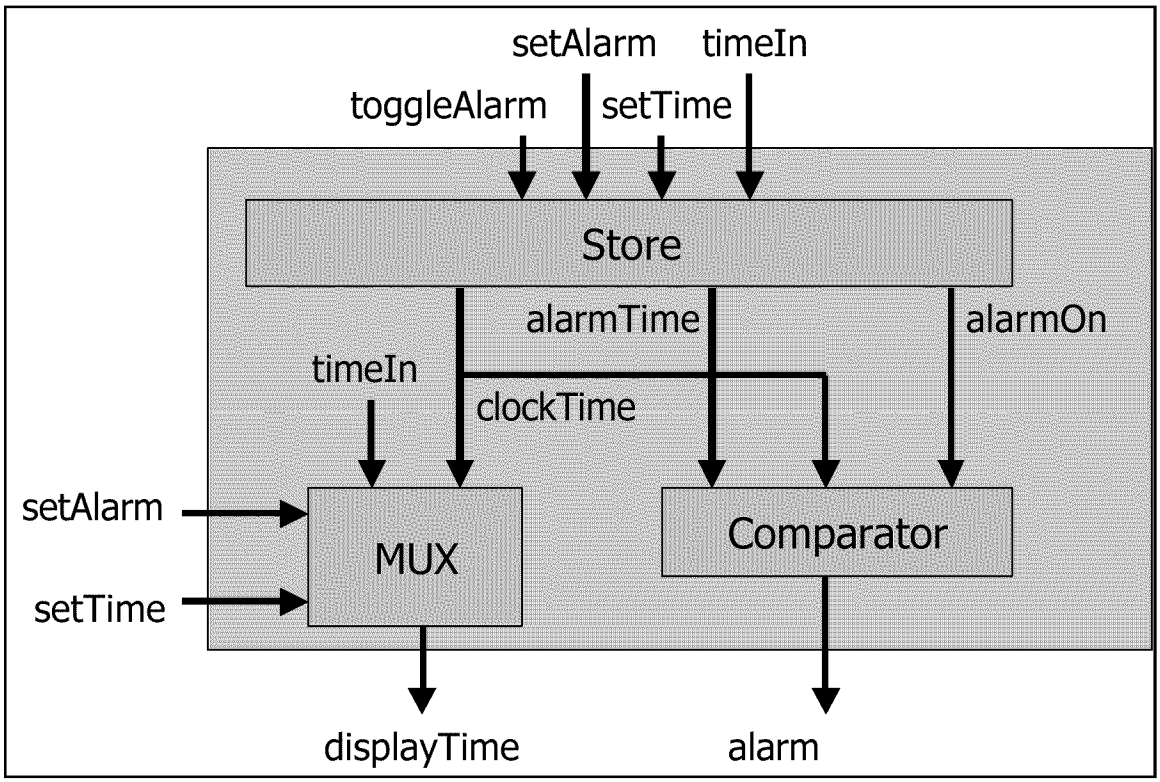

Figure 38. Alarm Clock Representation

In order to understand the code in the following sections, it would help to understand the function of the alarm clock. The primary functions are as follows:

- When the setTime bit is set, the timeIn is stored as the clockTime and output as the display time.
- When the setAlarm bit is set, the timeIn is stored as the alarmTime and output as the display time.
- When the alarmToggle bit is set, the alarmOn bit is toggled.

- When clockTime and alarmTime are equivalent and alarmOn is high, the alarm should be sounded. Otherwise it should not.
- When setTime is clear and setAlarm is clear, clockTime is output as the display time.
- The clock always increments its time value.

11.2 Alarm Clock Rosetta Specification

This section defines the following:
- Requirements specifications with timing constraints
- Component specification and structure
- Power and clock speed constraints
- Composite, heterogeneous, systems-level specification.

Before providing examples of the Rosetta code, the following term definitions are necessary:

timeTypes defines basic types and functions

alarmClockBeh defines real-time functional requirements
- Terms define functional requirements
- Hard real-time constraints specified using continuous time domain

alarmClockStruct defines the structure of an implementation

alarmClockConst defines a collection of constraints
- Individual facets describe separate constraints
- Conjunction used for facet assembly

When alarmClockLP defines a low power alarm clock
- Behavioral description asserts function and real-time constraints
- Structural description asserts a structure
- Constraints force a low power model
- Conjunction asserts all requirements simultaneously.

11.2.1 Example Alarm Clock Rosetta Code

The figures in this section show example code.

```
// Systems level specification of the overall alarm clock behavior
use timeTypes;
facet alarmClockBeh(timeIn::in time; displayTime::out time;
                    alarm::out bit; setAlarm::in bit;
                    setTime::in bit; alarmToggle::in bit) is
    alarmTime :: time; clockTime :: time; alarmOn :: bit;
begin continuous-time
    setclock: if %setTime
              then clockTime@t+5ms = timeIn
                   and displayTime@t+5ms = timeIn
              else clockTime@t+5 = clockTime endif;
    setalarm: if %setAlarm
              then alarmTime@t+5ms = timeIn
                   and displayTime@t+5ms = timeIn
              else alarmTime@t+5ms = alarmTime endif;
    tick: clockTime@t+5ms = increment-time clockTime;
    armalarm: if %alarmToggle
              then alarmOn@t+5ms = -alarmOn
              else alarmOn@t+5ms = alarmOn endif;
    sound: alarm@t+5ms = alarmOn and %(alarmTime=clockTime);
 end alarmClockBeh;
```

Figure 39. Systems Level Alarm Clock

```
// Basic times used for component and system specifications
package timeTypes is
  begin logic
    hour :: type(natural) is sel(x::natural | x =< 12);
    minute :: type(natural) is sel(x::natural | x =< 59);
    time :: type(univ) is record [h::hours | m::minutes];

    increment_time(t::time)::time is
       record[h = increment_hours(t) | m = increment_minutes(t)];

    increment_minutes(t:: time)::minute is
        if t(m) < 59 then t(m)+1 else 0 endif;

    increment_hours(t:: time)::hours is
        if t(m) = 59 then if t(h) < 12
                             then t(h) + 1
                             else 0 endif
                     else t(h) endif;
 end timeTypes;
```

Figure 40. Time Types Package

```
// MUX selects from among inputs based on operation performed
use timeTypes;
facet mux(timeIn::in time; displayTime::out time;
      clockTime::in time; setAlarm::in bit; setTime::in bit) is
  begin state-based
    l1: if %setAlarm then alarmTime' = timeIn endif;
    l2: if %setTime then displayTime' = timeIn endif;
    l3: if %(-(setTime or setAlarm)) then displayTime'=clockTime endif;
end mux;
```

Figure 41. Display Multiplexer

```
// Storage for alarm clock information including clock time, alarm time
     and
// alarm on/off state
use timeTypes;
facet store(timeIn::in time; setAlarm::in bit; setTime::in bit;
          toggleAlarm::in bit; clockTime::out time;
          alarmTime::out time; alarmOn::out bit) is
begin state_based
  l1: alarmTime'=if %setAlarm then timeIn
                       else alarmTime endif;
  l2: clockTime'=if %setTime then timeIn
                       else increment_time(clockTime) endif;
  l3: alarmOn'=if %toggleAlarm then -alarmOn
                       else alarmOn endif;
end store;
```

Figure 42. Alarm State Storage

```
// Compares current time with alarm time and sets the alarm output
// indicator
use timeTypes;
facet comparator(alarmOn:: in bit; alarmTime:: in time;
              clockTime:: in time; alarm:: out bit) is
begin state-based
   l1: alarm' = %(alarmOn and (alarmTime=clockTime))
end comparator;
```

Figure 43. Comparator

```
// Structural definition combining store, comparator, and MUX
     definitions
use timeTypes;
facet alarmClockStruct(timeIn::in time; displayTime::out time;
                  alarm::out bit; setAlarm::in bit; setTime::in bit;
                  alarmToggle::in bit) is
  clockTime :: time;
  alarmTime :: time;
  alarmOn :: bit;
begin logic
  store_1 : store(timeIn,setAlarm,setTime,alarmToggle,clockTime,
                  alarmTime,alarmOn);
  comparator_1 : comparator(setAlarm,alarmTime,clockTime,alarm);
  mux_1 : mux(timeIn,displayTime,clockTime,setAlarm,setTime);
end alarmClockStruct;
```

Figure 44. Structural Definition

```
// Alarm clock power constraints
facet alarmClockPower(lp::design boolean) is
  p::power;
begin constraints
  pwr: if lp then p=<10mW else p=<40mW endif;
end alarmClockPower;

// Alarm clock clockspeed constraints
facet alarmClockClkspd is
  clockspeed::frequency;
begin constraints
  clk: clockspeek =< 1MHz;
end alarmClockClkspd;

// Combined constraints specifications
alarmClockConst(lp::design boolean)::facet =
   alarmClockPower(lp) and alarmClockClkspd;
```

Figure 45. Constraints Definitions

```
// Low Power alarm clock
alarmClockLP :: facet is alarmClockStruct and alarmClockBeh
   and alarmClockConst(TRUE);

// Variable mode alarm clock
alarmClock(pm::design boolean) :: facet is
   alarmClockStruct and
   alarmClockBeh and
   and alarmClockConst(pm);
```

Figure 46. Low Power and Mixed Specification

11.2.2 Alarm Clock Test Facet

```
FACET alarmClockBeh_TEST( timeIn:: in time; displayTime:: out time; alarm::
    out bit; setAlarm:: in bit; setTime:: in bit; alarmToggle:: in bit)  IS
alarmTime :: time;
clockTime :: time;
alarmOn :: bit;
BEGIN state_based
  ACCEPT1_setclock:  ( setTime = 1 )  AND ( clockTime' = timeIn ) AND (
    displayTime' = timeIn );
  ACCEPT2_setclock:  ( setTime = 0 )  AND ( clockTime' = clockTime );
  ACCEPT1_setalarm:  ( setAlarm = 1 )  AND ( alarmTime' = timeIn ) AND (
    displayTime' = timeIn );
  ACCEPT2_setalarm:  ( setAlarm = 0 )  AND ( alarmTime' = alarmTime );
  ACCEPT1_displayClock:  ( setTime = 0  ) AND ( setAlarm = 0  ) AND (
    displayTime' = clockTime );
  ACCEPT1_tick:  ( clockTime' = increment_time(clockTime) );
  ACCEPT1_armalarm:  ( alarmToggle = 1 )  AND ( alarmOn' = -alarmOn ) ;
  ACCEPT2_armalarm:  ( alarmToggle = 0 )  AND ( alarmOn' = alarmOn ) ;
  ACCEPT1_sound:  ( alarm' = alarmOn ) AND ( alarmTime = clockTime );
END alarmClockBeh_TEST;
```

Figure 47. Example Test Facet

11.3 Behavioral Alarm Clock VHDL Code

The figures in this section show example VHDL code for the behavioral alarm clock.

```
entity ALARM_BLOCK is
      port (ALARM,HRS,MINS,CLK: in BIT;
            CONNECT9:buffer INTEGER range 1 to 12;
            CONNECT10: buffer INTEGER range 0 to 59;
            CONNECT11: buffer BIT);
end;

architecture BEHAVIOR of ALARM_BLOCK is

      component ALARM_STATE_MACHINE
      port(ALARM_BUTTON: in BIT;
           HOURS_BUTTON: in BIT;
           MINUTES_BUTTON: in BIT;
           CLK:in BIT;
           HOURS: out BIT;
           MINS: out BIT);
      end component;

      component ALARM_COUNTER
      port (HOURS: in BIT;
            MINS: in BIT;
            CLK: in BIT;
            HOURS_OUT: buffer INTEGER range 0 to 12;
            MINUTES_OUT: buffer INTEGER range 0 to 59;
            AM_PM_OUT: buffer BIT);
      end component;

--  Top level nets that connect major modules

      signal CONNECT1,CONNECT2 : BIT;

      begin
         U1: ALARM_STATE_MACHINE port map
             (ALARM,HRS,MINS,CLK,CONNECT1,CONNECT2);
         U2: ALARM_COUNTER port map
             (CONNECT1,CONNECT2,CLK,CONNECT9,CONNECT10,CONNECT11);
      end;
```

Figure 48. ALARM_BLOCK.vhd

```
entity ALARM_STATE_MACHINE is
   port (ALARM_BUTTON, HOURS_BUTTON, MINUTES_BUTTON, CLK: in BIT;
        HOURS, MINS: out BIT);
end;

architecture BEHAVIOR of ALARM_STATE_MACHINE is
   type STATE_TYPE is (IDLE,SET_HOURS,SET_MINUTES);
   signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
begin

 COMBIN: process(CURRENT_STATE, ALARM_BUTTON, HOURS_BUTTON, MINUTES_BUTTON)
 begin
  NEXT_STATE <= CURRENT_STATE;
  HOURS <= '0';
  MINS <= '0';
  case CURRENT_STATE is
   when IDLE =>
          if (ALARM_BUTTON  = '1' and HOURS_BUTTON  = '1' and MINUTES_BUTTON = '0') then
            NEXT_STATE <= SET_HOURS;
            HOURS <= '1';
          elsif (ALARM_BUTTON  = '1' and MINUTES_BUTTON  = '1' and HOURS_BUTTON = '0') then
            NEXT_STATE <= SET_MINUTES;
            MINS <= '1';
          else
            NEXT_STATE <= IDLE;
          end if;
    when SET_HOURS =>
          if (ALARM_BUTTON  = '1' and HOURS_BUTTON  = '1'  and MINUTES_BUTTON = '0')then
            NEXT_STATE <= SET_HOURS;
            HOURS <= '0';
          else
            NEXT_STATE <= IDLE;
          end if;
    when SET_MINUTES =>
          if (ALARM_BUTTON  = '1' and MINUTES_BUTTON  = '1'  and HOURS_BUTTON = '0') then
            NEXT_STATE <= SET_MINUTES;
            MINS <= '0';
          else
            NEXT_STATE <= IDLE;
          end if;
   end case;
  end process;

  SYNCH: process
  begin
   wait until CLK'event and CLK = '1';
   CURRENT_STATE <= NEXT_STATE;
  end process;

end BEHAVIOR;
```

Figure 49. ALARM_STATE_MACHINE.vhd

```
entity ALARM_COUNTER is
   port (HOURS, MINS, CLK : in BIT;
            HOURS_OUT : buffer INTEGER range 1 to 12 := 12;
            MINUTES_OUT : buffer INTEGER range 0 to 59 := 0;
            AM_PM_OUT: buffer BIT:= '0');
end;

architecture BEHAVIOR of ALARM_COUNTER is
begin

  process
  begin
   wait until CLK'event and CLK = '1';

          MINUTES_OUT <= MINUTES_OUT;
          HOURS_OUT <= HOURS_OUT;
          AM_PM_OUT <= AM_PM_OUT;

          if (MINS = '1' and HOURS = '0') then
            if MINUTES_OUT = 59 then
              MINUTES_OUT <= 0;
              if HOURS_OUT = 12 then
                    HOURS_OUT <= 1;
                    AM_PM_OUT <= not AM_PM_OUT;
              else
                    HOURS_OUT <= HOURS_OUT + 1;
              end if;
            else
              MINUTES_OUT <= MINUTES_OUT + 1;
            end if;
          elsif (HOURS = '1' and MINS = '0') then
            if HOURS_OUT = 12 then
              HOURS_OUT <= 1;
              AM_PM_OUT <= not AM_PM_OUT;
            else
              HOURS_OUT <= HOURS_OUT + 1;
            end if;
          end if;

  end process;

end BEHAVIOR;
```

Figure 50. ALARM_COUNTER.vhd